4-2020

A Study on DNA Memory Encoding Architecture

Robert W. Mason
rm1695@rit.edu

A Study on DNA Memory Encoding Architecture

Robert W. Mason

A Study on DNA Memory Encoding Architecture

Robert W. Mason

April 2020

A Thesis Submitted
in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in
Computer Engineering

R·I·T | Kate Gleason
College of ENGINEERING

Department of Computer Engineering

A Study on DNA Memory Encoding Architecture

Robert W. Mason

Committee Approval:

Dr. Santosh Kurinec *Advisor* Date
Professor

Dr. Cory Merkel Date
Assistant Professor

Dr. Sonia Lopez Alarcon Date
Associate Professor

Dr. Michael Savka Date
Professor

Acknowledgments

I wish to thank all of my professors and committee here at RIT for the amazing support that they've provided me throughout the years. I also wish to thank Dr. Moon for pushing me to always continue learning. I also wish to thank Adam Hieb and Dr. John Randall for all of the motivation to keep working on what I'm passionate about.

I want to thank my parents and brother for putting up with me throughout the entire research process.

But most of all, I want to thank Dr. Kurinec for all of the help over the past few years. Whether it be lectures, interesting reading, or even various conversations, it has been a pleasure working with her.

Abstract

The amount of raw generated data is growing at an exponential rate due to the greatly increasing number of sensors in electronic systems. While the majority of this data is never used, it is often kept for cases such as failure analysis. As such, archival memory storage, where data can be stored at an extremely high density at the cost of read latency, is becoming more popular than ever for long term storage. In biological organisms, Deoxyribonucleic Acid (DNA) is used as a method of storing information in terms of simple building blocks, as to allow for larger and more complicated structures in a density much higher than can currently be realized on modern memory devices. Given the ability for organisms to store this information in a set of four bases for an extremely long amounts of time with limited degradation, DNA presents itself as a possible way to store data in a manner similar to binary data. This work investigates the use of DNA strands as a storage regime, where system-level data is translated into an efficient encoding to minimize base pair errors both at a local level and at the chain level. An encoding method using a Bose-Chaudhuri-Hocquenghem (BCH) pre-coded Raptor scheme is implemented in conjunction with an 8 to 6 binary to base translation, yielding an informational density of 1.18 bits/base pair. A Field-Programmable Gate Array (FPGA) is then used in conjunction with a soft-core processor to verify address and key translation abilities, providing strong support that a strand-pool DNA model is reasonable for archival storage.

Contents

List of Figures

List of Tables

List of Acronyms

3D Three Dimensional

ASCII American Standard Code for Information Interchange

AWS Amazon Web Services

BCH Bose-Chaudhuri-Hocquenghem

bp base pair

CHEMFET Chemical Field Effect Transistor

CMOS Complimentary Metal-Oxide-Semiconductor

DARPA Defence Advanced Research Projects Agency

DNA Deoxyribonucleic Acid

DWM Domain Wall Memory

ECC Error-Correction Codes

EEPROM Electrically Erasable Programmable Read-Only Memory

FeRAM Ferroelectric RAM

FG Floating-Gate

GF(x) Galois Field of dimension x

HAMR Heat-Assisted Magnetic Recording

HDD Hard Disk Drive

HDMR Heated Dot Magnetic Recording

LOC Lab-on-a-chip

MRAM Magnetic RAM

NAND Not-AND

PCO Pre-Code-Only

PCR Polymerase Chain Reaction

PCRAM Phase-Change RAM

PRT Pool Recall Time

RAM Random Access Memory

RNA Ribonucleic Acid

RPM Rotations Per Minute

RS Reed-Solomon

SAS Serial-Attached SCSI

SLC Single-Level Cell

SNP Single-Nucleotide Polymorphism

SSD Solid-State Drive

VHDL Very High Speed Integrated Circuit Hardware Description Language

Chapter 1

DNA as a Storage Method

This chapter targets an understanding of archival memory as a whole, and how Deoxyribonucleic Acid DNA can be used practically to fill this need. In particular, the different existing methods for archival storage are developed, and basic DNA theory is explored.

1.1 A Practical Understanding of Memory

As a basis, the concept of computer memory revolves around some structure that contains a characteristic signature, such that a clear differentiation can be made between an effective on and off state. For situations where one wished to store data for a longer period of time, early non-volatile memory consisted of punch-cards, where binary data was represented through a physical hole in some kind of material[1]. As more memory was needed, this gave way to punched tape, and later the first magnetic memory. Later, the emergence of the first hard drive (HDD), marked a new age in storage density due the extremely high storage density.

However, with the advent of the first floating gate type memory solutions, based on the concept of consistently re-programmable EEPROM, a major divide in memory storage was made[2]. Combining the non-volatility and access speed of EEPROM with the device lifetime of modern transistor devices, a solid-state solution to memory was found, at the cost of physical density. In the case of hard-drives, it is much easier to

achieve a high storage density than NAND-based Solid State Drives (SSDs), which required both a full floating-gate (FG) transistor design as well as significant read and write overhead. Thus, depending on the speed and storage requirements, it became necessary to choose the non-volatile storage solution as opposed to the previous catch-all devices. Unlike that of HDDs, the floating-gate design allowed electron migration, leading to much higher bit-rate errors and the popularization of Error-Correction Codes (ECC). For example, the 2013 enterprise SAS specification for HDDs limited errors to 1 in 10^{16} bits, while modern SSDs can be as bad as 1 in 10^{6} for modern 4-bit per cell NAND[3].

At the same time, the purpose of electronic memory began to shift between active useful data and unprocessed data. In large corporations, automated testing began to reach a point where it was no longer able to process all of the information produced, but instead it became necessary to store all of it in the event that it was later needed. Over the past two decades, the amount of data generated by both companies and users has exploded, leading to an ever increasing need for high density storage with strict read and write requirements. With some figures estimating 40 ZB of data to be generated in 2020, it has become virtually impossible for all information to be fully processed, let alone understood[4]. For these cases where a single read operation isn't even guaranteed, the access times become a non-factor, making HDDs and other slower devices useful once again.

While NAND SSDs are fast and dense enough for most situations due to modern improvements in vertical NAND arrays, this comes at a higher leakage current and a much lower device lifetime. In particular, modern charge-trap NAND devices suffer from memory drift, where data must be refreshed every few months to prevent mis-reads[5], severely limiting the ability to act as a proper archival storage. Due to the trends of wanting faster access time at the cost of operation, companies are moving back to tape-based devices for their archival needs or instead low-RPM HDDs[6].

With these quickly approaching their physical density limits, it is more and more necessary to explore non-conventional methods for storing this archival data.

Memory in modern server systems can be broken down into the commonly referenced hierarchy of memory. A modified version of this pyramid is shown below. Obviously, for small memory types, they are closer to the processor itself and are accessed at a much higher frequency, increasing the power cost but providing significantly better response times. At the bottom, memory is extremely slow, but is orders of magnitude larger in size. It is noted that archival storage in this figure is broken into a traditional archival storage and deep-archival, which is meant to represent the needs for 'store-once' type architectures.

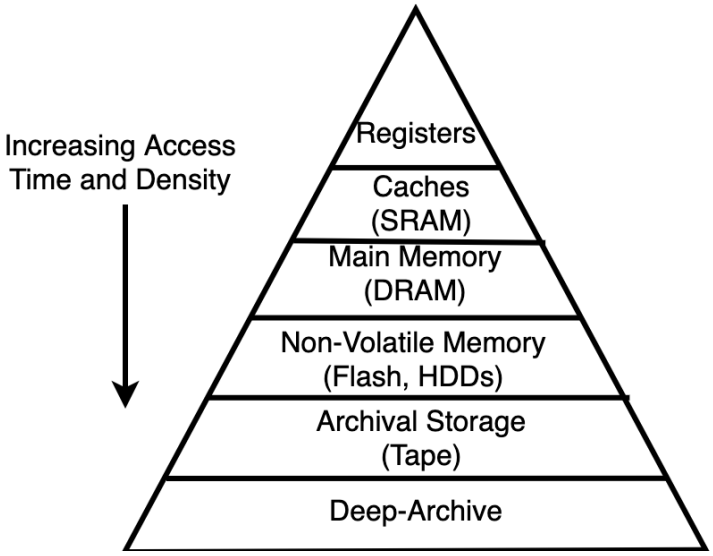

Figure 1.1: Depiction of the hierarchy of memory showing that as read latency increases, the memory size increases.

In particular, at the lowest level of the pyramid, that of deep-archival, there are very few proposed memory systems due to the novelty of computational need. Deep-archival in this respect is in reference to all data that should be stored but is not expected to be read at all. One commonly suggested option is the idea of using the Deoxyribonucleic Acid bases found in living organisms due to their logical parallels

to electronic memory.

1.2 DNA as a Memory Source

DNA has long been investigated as both an archival memory solution as well as a computational method due to the similarities with binary memory from the chain linearity and binary-equivalent blocks[7]. Formed from two connecting chains in a double-helix spiral, double-stranded DNA (dsDNA) as a linear polymer is composed of four building blocks, Adenine, Guanine, Cytosine, and Thymine, which bond to their opposite, A-T and G-C along a sugar-based backbone. These chains are direction-based with one end being referred to as the 5' end while the other is the 3', which defines the antiparallel nature of a dsDNA system. A simple DNA strand is shown in Fig. 1.2.

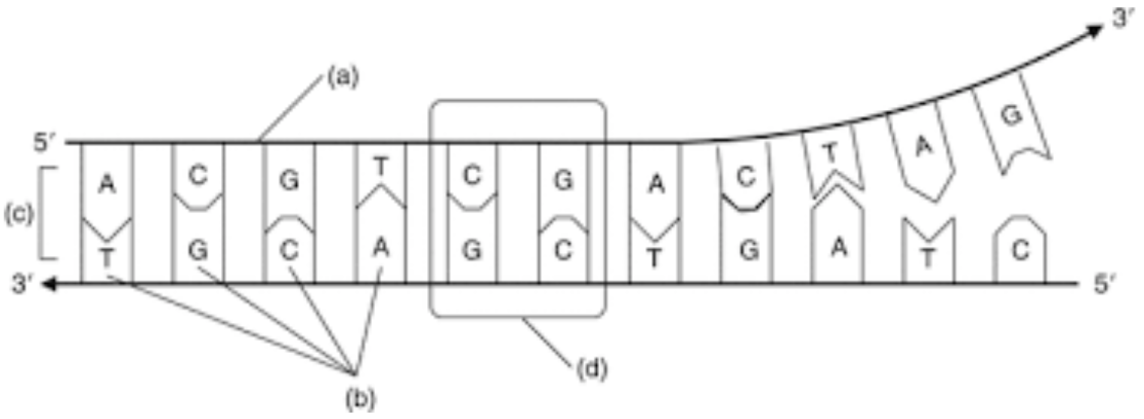

Figure 1.2: Depiction of a DNA strand showing the backbone (a), the four base pairs (b), base pair bonding to form the double helix (c), and bond reversal (d). [7].

Furthermore, unlike most other theoretical memory technologies, there is proof of high storage density already in existence, through all living organisms. In a perfect world, potentially two bits could be stored in each mer unit, however other factors prevent this reality and greatly reduce efficiency. In the biological realm, the vast majority of data in DNA is only indirectly useful, so it raises the question of whether an archival system would have these same inefficiencies.

The application of DNA, specifically through the use of four base pairs (A,C,G,T) as a substitute for data bits, present themselves as an obvious solution for storage due to their incredible physical density and estimated half life of 520 years[8]. Ignoring peripherals and assuming this perfect encoding of two bits per base, this provides a theoretical density of 1EB/mm^3[9]. Before this becomes a commercial reality however, a large number of issues from an architecture perspective need to be solved[10].

In particular, with entities such as DARPA investing more than 50 million dollars in the next few years into various projects, there is a distinct need for a unified system example for a DNA compute unit[11]. Seeing how DNA memory is a non-traditional storage, it becomes necessary to develop a new form of address translation, in the sense of computer architecture rather than the traditional genetic sense, as well as a new methodology to describe a temporarily full system, which is unlike Solid State Drive (SSD) single-bit cell (SLC) caching[12]. In the case of SLC caching slowdowns, this is a comparatively minor hit as opposed to a change from NAND, the common storage format of modern SSDs, folding to a complete DNA chemical synthesis process[13, 14]. Due to this, a new architecture for write misses needs to be developed in order to significantly reduce system slowdowns.

1.3 Basics of Strand Errors

In traditional electronic memory, there are well-defined sources of error, due to events such as magnetic bit-flips, electron migration, or various other understood phenomena. DNA however behaves differently due to a much higher number of error cases. Instead of simply having more or fewer electrons than expected in a read, errors are much more binary in nature, where a base value is either correct or incorrect. However, due to the nature of bases themselves, one error generally has a direct effect on the surrounding bases making error sequences more likely. Below, a number of common DNA chain errors are briefly discussed, both as to how they are formed and

how likely they are to appear.

It is important to note a distinction with DNA memory when compared to biological mutations. Specifically, the availability of base pairs in new strand synthesis is a known factor, where the injected leading to a probabilistic argument as to the possibility of specific mutation occurrences. Additionally, the use of the term mutation in this context is the discussion of short DNA chain mutations, not larger chromosomal-type. Heredity mutation is also ignored because of the single-replication nature of this proposed memory.

1.3.1 Single-base Substitution

The logical starting point when discussing base mismatches is that of a single-base substitution, which is an electronic analog to a bit-flip. This is not an exact parallel to Single-Nucleotide Polymorphism (SNP), because there is no formation of the reference amino acid, and thus a non-functioning protein would not be created across copies but is more akin to a point mutation[15]. An example of this would be an AT bonding which instead turned into an TA.

Due to this possibility, it becomes necessary to build a model to predict what the frequency of these bit errors are, particularly because they occur much higher in frequency than HDDs would find acceptable at 10^{-10}[16]. Even further, there is no ability to perform true mismatch repair, so it is assumed that all sequencing mismatches will remain in storage[17]. By building more accurate models parallel to the conditions in the sequencing chamber, it becomes more realistic to predict both the likelihood of point errors based on the surrounding sequence, but also get a better idea of what errors are readily correctable. This provides an advantage over regular memory, which generally is unable to provide useful recovery data from surrounding bit cells.

1.3.2 Frameshift Mutations

Furthermore, deletions and insertions also known as indels, which are the lack of a base or the addition of an extra base leading to a frameshift, are also possible[18]. These are generally more likely to occur in short repeated sequences, which gives rise to one of many general rules for encoding sequences. These are significantly more abstract from the perspective of electronics. In most literature, they are concerned with the shift in reading frame and stop codons, which are much less important in electronics. However, in traditional memory, there is no ability for an extra bit of information to appear, thus for a DNA system, it becomes necessary to correct for this in a non-traditional manner.

By looking at the probability of such an insertion, its possible to establish the maximum number of insertions that would be likely in an even such as this and read the decoded sequence into a buffer of that length. The microcontroller would then have to perform some degree of error correction to understand both where the insertion likely occurred, as well as how much of the data can be corrected. In theory, this could be handle directly in decode logic, but likely microcontroller ECC flow would be tasked with this. In living organisms however, a proof-reading mechanism can correct some mutations enzymatically using the complementary strand of the dsDNA. In a truly biological compute system this could be leveraged, but at the present this is deemed out of realistic scope.

1.3.3 Hairpin Loops

One final important synthesis error is through the creation of hairpin errors, another example of a non-electrical type issue. These generally occur when regions of a strand have complimentary sequences in opposite directions which self-bond. While useful for many RNA secondary structures, these are undesirable for long-term DNA storage.

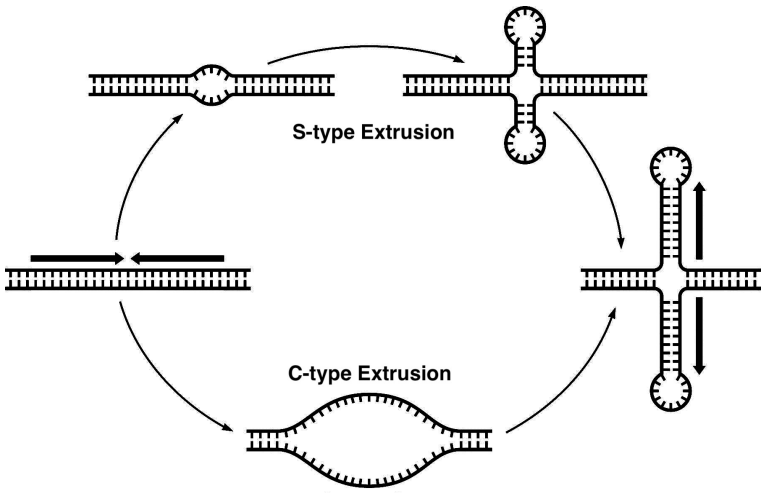

Figure 1.3: Demonstration of how different extrusions can lead to hairpin loops.[19]

While these errors aren't particularly common, it is important to highlight that they do exist, and that new methods of error correction need to be developed.

1.4 Understanding Synthesis

A logical follow up onto all of the major DNA base errors is the discussion of synthesis and sequencing itself.

1.4.1 Synthesis

One of the major barriers to feasibility with DNA memory is the extreme cost of accurate synthetic DNA synthesis, which acts as an analog to the write in modern computing. It is unnecessary to fully develop an understanding of all methods for DNA synthesis, so instead only column-based oligonucleotide synthesis and micro-array based synthesis are explored. While *de Novo* has been shown to be more effective, the prohibitively high costs for this application make it unlikely to act as a solution[20].

The concept of column-based oligonucleotide synthesis is relatively simple, hinging on a four step elongation cycle to slowly build individual DNA strands[21]. This

synthesis method is reasonably accurate, with an error rate around 0.5%, with yields around 99%. For a continuous drive system this would have to be improved, but is a reasonable starting point.

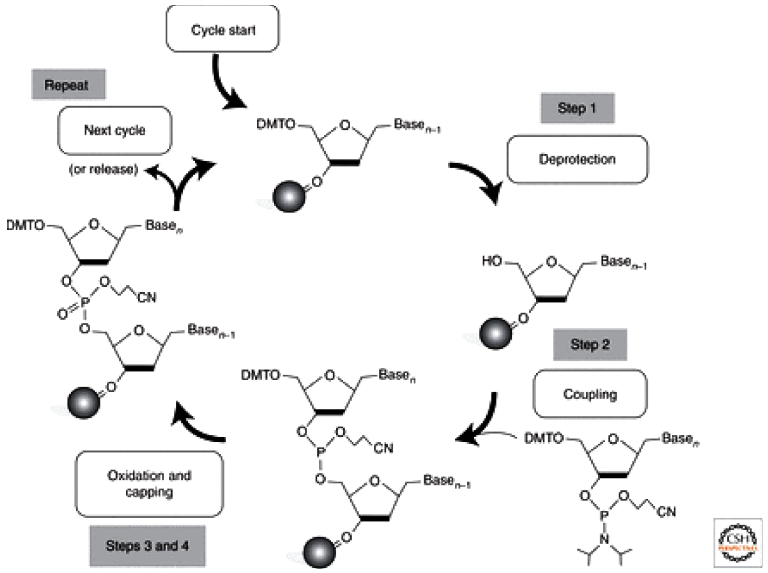

Figure 1.4: Depiction of four step cycle of chain elongation. Step 1 is a de-protection using an acid. Step 2 adds the new base to the chain. Step 3 caps the chain to prevent extra growth. Step 4 properly links the monomer to the backbone.[21]

However, due to the still relatively high cost per bp, this is still prohibitively expensive leading to the need for micro-array based solutions, which are significantly cheaper, but come at the cost of many more errors. However, this should be correctable through ECC methods, so depending on the actual error rate, this is potentially acceptable. In the next few years it is expected for synthesis to become much more stable, removing another technological barrier. Furthermore, with significant work being done on microfluidic systems for DNA synthetic synthesis, a complete lab on chip package may be reasonable to expect[22].

1.4.2 Sequencing

Given that we have the ability to directly control the total chain length of each DNA sequence for this memory type, it is a logical step to make the decision to only work

with shorter strand lengths, such that Shotgun Sequencing is unnecessary[23]. By sticking to shorter strands, it is possible to ideally read the entirety of each memory chain in one attempt. Shorter strands can be sequenced, however they require overlap to chain together, which would impact the efficiency in a memory system.

Sequencing technology can be broken up into three distinct methods, First Generation, Next-Generation, and Third-Generation. Beginning with First Generation sequencing, originally developed as Sanger Sequencing in 1977, this provided some of the earliest methods of sequencing DNA strands. At a high level, this process uses the Chain-Termination method, which provides an ability to read a single base along a chain at a time in an out of order sequence. The figure below outlines the basic process for Sanger Sequencing.

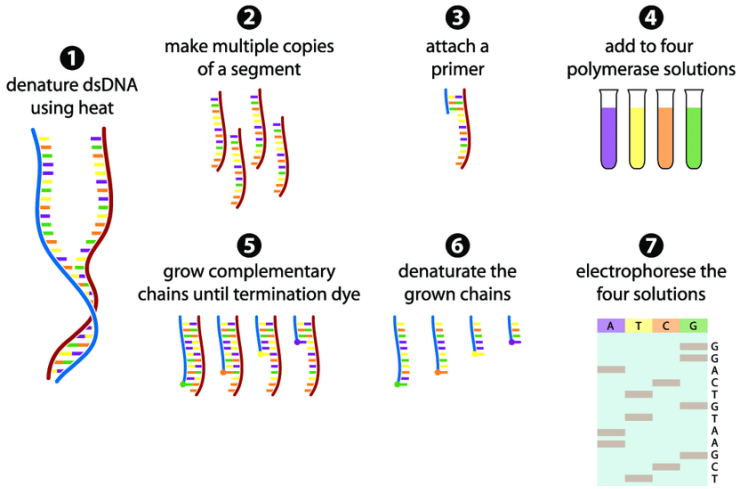

Figure 1.5: Depiction of the Sanger Sequencing method. 1-3 demonstrate chain separation and replication, followed by addition to specific polymerase solutions where chains are grown in 5 and separated in 6. 7 then shows the resulting electrophorese to reconstruct the sequence.[24]

With an extremely high (greater than 99.9999%) accuracy and the ability to read chains up to 1000 base pairs (bp) long, this seems like a viable solution. Unfortunately, this is an extremely slow and costly process, which makes it harder to argue the case for DNA memory. This can then be read in numerous ways to provide the binary base representations.

More recently in 2005, sequencing moved to Next-Generation Sequencing, which was primarily focused around cyclic-array methods[25]. DNA strands are fragmented and specific adapters, which are artificial DNA sequences are added to create a sequencing library. These are then cyclically decoded to rapidly sequence the entire strand[26].

While this method has a much higher throughput due to the simultaneous sequencing ability, it comes at a cost of a much higher error rate, as well as a much more complicated set of errors. However, this error rate of 1-2.5% is still correctable using modern decoding methods. Unfortunately, these reads have to be done on shorter strands, within a limit of a few hundred bp instead. This is still possible, but places a greater pressure on synthesis and will ultimately impact the bit efficiency of the data. Unfortunately, unlike Sanger Sequencing, these errors are substitution errors towards the end of reads and after CG groups. These sequences also will sometimes contain adapter errors, which requires an additional computational step to accurately remove. Additionally, the library preparation step can often introduce errors due to its own reliance on polymerase chain reaction (PCR).

The final, and most recent sequencing is that of Third-Generation sequencing, which rely on single-molecule sequencing directly, rather than relying on multiple replicated copies[27]. This makes the read operation significantly faster due to the removal of extra chemistry steps. This process is done either through direct real-time sequencing or through nanopore webbing to identify individual bases at a time. The figure below shows a cross-section of one such nanopore design, where decoding is slightly more clear.

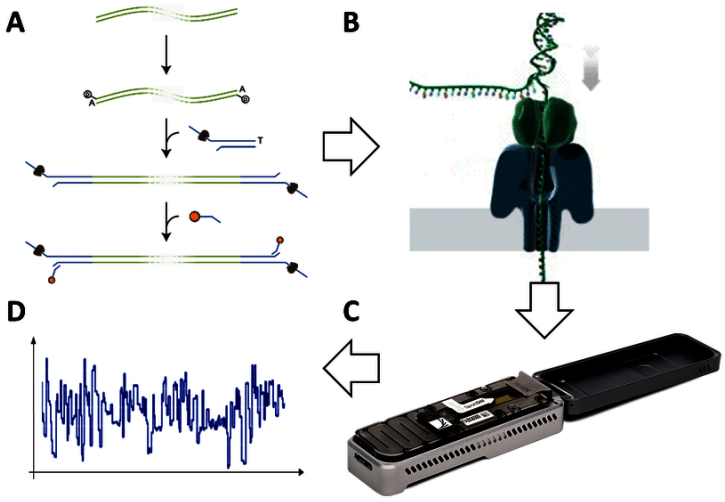

Figure 1.6: Image depicting (A) chain separation, (B) pass through a nanopore collector setup, (C) electrical collection of voltage data, and (D) output intensity of base strands.[28]

These methods have the ability to handle much longer chains (into the thousands) and much cheaper reads, which is ideal for DNA memory. While the error rates are extremely high, 15-30%, it is clear that this will drop as these technologies become more mature. Comparing these methods, it is clear the Third-Generation sequencing will eventually be the method used for DNA memory assuming that the expected improvements come true. Particularly, this method appears to have the highest likelihood for lab on chip designs to potentially create a more small form factor DNA memory drive. In the meantime, it seems likely that errors can be handled by simply collecting multiple copies of the same information strand to perform accurate reads. This is a concern due to the destructive nature of reads, making single try systems important.

1.4.3 Polymerase Chain Reactions

Much of these sequencing tools make large use of polymerase chain reactions (PCR) to make large numbers of copies, typically in the 10^6 to 10^9 range through the repeated use of a DNA polymerase enzyme such as Thermus aquaticus (Taq) polymerase[29].

The existing DNA strands are treated as templates to allow for the creation of new copies. The number of copies then increases by powers of two based on the number of PCR cycles. This comes at the cost of larger processing time, so the goal for DNA memory is to sequence with the smallest sample size possible. The basic process of PCR is outlined in Figure 1.7.

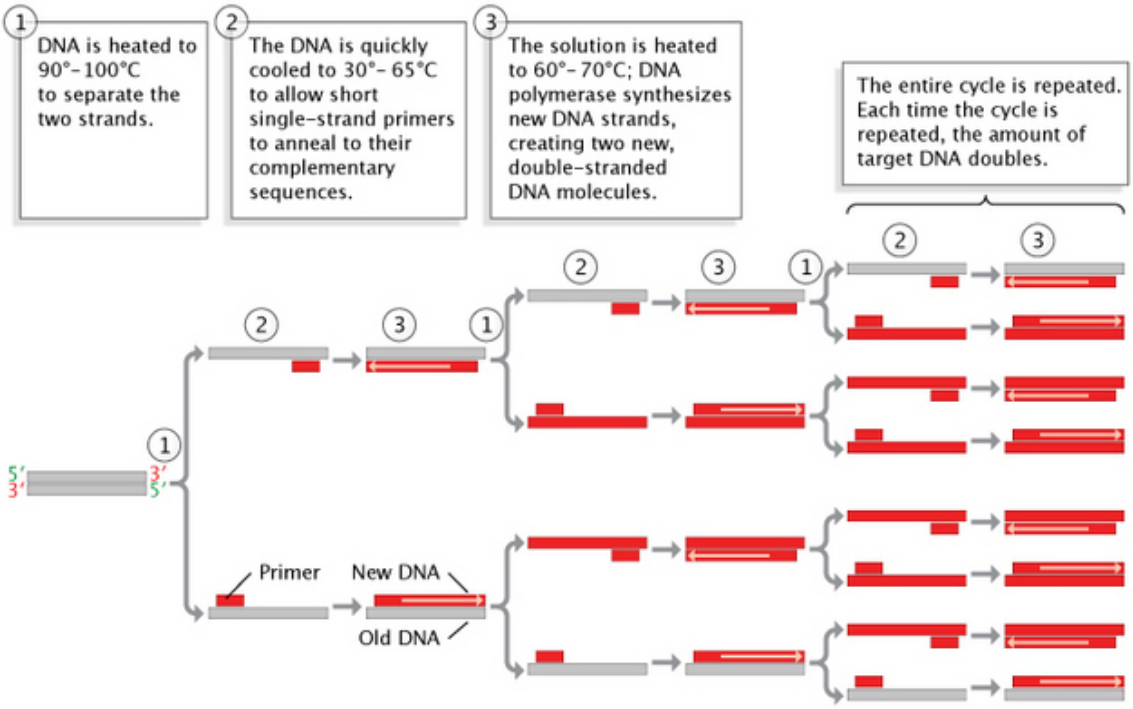

Figure 1.7: Figure depicting the basic process of PCR.[30]

The basic cycle of the PCR process begins by the denaturation step, which is a heating of the dsDNA strands to separate them into the complimentary ssDNA, providing the templates. The second step then uses specific primers, which bond onto the complimentary sequences on the ssDNA. The Taq polymerase then extends these primers to make the complimentary strands, creating two copies of the dsDNA from the original. This process is then repeated over and over until a large enough sample is generated.

1.4.4 Micro-electromechanical DNA Systems

One other major concern to be noted is that most modern works assume read and write operations on DNA memory to be performed in full-scale sequencing tools rather than on a Microelectromechanical System (MEMS) design, leading to a split between the proposed full lab-on-chip (LOC) setup versus a much larger write and store system, where the actual synthesis is done on a much larger system. While much bulkier, this would allow for increased precision at the cost of speed and power. Read latency once again is hurt but largely irrelevant due to the astronomical requirements already.

Microsoft in particular provides work that suggests that a full standalone chip will be developed in the future through PUDDLE[31]. This work focuses on the full-stack automation of microfluidic LOCs rather than the chip design itself, but provides insight into where they expect DNA memory to go in the future. The target is to manage low-cost droplet microfluidic setups such that they can be controlled computationally. In particular, they verify operation through the automation of Polymerase Chain Reaction(PCR) operations and a DNA sequencing protocol, which is directly applicable to DNA memory. The figure below demonstrates their proposed firmware setup.

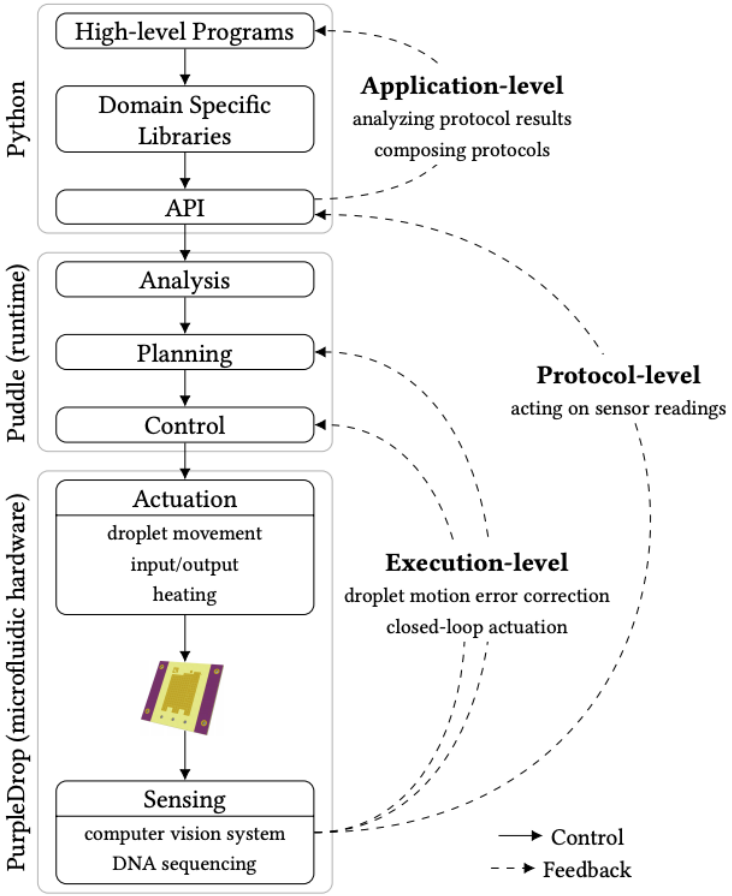

Figure 1.8: Depiction of the PUDDLE firmware setup, beginning with a high level Python control scheme, which is then translated into actions for the PurpleDrop microfluidic hardware. [31].

This provides an early example of how a DNA memory system could be converted into a microcontroller operational scheme. By offloading the PUDDLE translation onto an on-board microcontroller, it is easy to see how this could be broken into a storage system. The front end would handle the actual data storage, while the back end would control the microfluidic operations similar to how modern microcontrollers control individual NAND die, which are the specific smaller memory components that make up an SSD.

This then leads to a natural discussion of the trade off between a longer strand length versus a shorter strand length. From a purely spacial perspective, longer

strands are better due to their higher density, however due to the constraints of moderns synthesis tools, a MEMS approach is necessary. In modern works, error-free synthesis often takes in excess of 10 seconds per base once all portions of synthesis are complete[32]. At that rate, to write even a GB of data, it is prohibitively costly in terms of time. The vast parallelization using MEMS devices is the likely solution, where large numbers of strands are sequenced at the same time. This comes with a natural trade-off of higher error probabilities and numerous design constraints. The main key then becomes the "at write" correction of strands, or more specifically the rejection of poorly synthesized strands before storage. Smaller strands are then seen as being more feasible in the near term due to the lower error likelihood, while longer strands will eventually be targeted once synthesis is no longer the bottleneck.

1.5 Other Archival Storage Methods

In addition to the proposal of DNA memory as an archival replacement, there are a number of other existing options that it must be compared with. Ignoring theoretical universal memory solutions, current archival methods include hard drives and magnetic tape, with numerous other emerging memory solutions also being proposed.

1.5.1 Hard Drives

While HHDs were phased out in data servers for the numerous benefits provided by SSDs, they are seeing a return in the form of low RPM longer storage drives. In particular, Amazon Web Services (AWS) provides a product in the form of their cold HDD, which makes use of extremely slow spinning HDDs for low data access applications, making it ideal for deep archival storage[33]. For archival storage, it is paramount to minimize power usage, particularly idle power usage. If this technology is to continue along this development, there will be even slower drives in the near future.

With more modern advances in HHDs, there is a clear push towards increasing storage density at the cost of access time. However, similar to flash memory, there is a limit on horizontal footprint cell scaling, but there isn't a parallel to the 3D vertical integration that NAND has seen. As drives move ever closer to the Slater-Pauling limit, Heat-Assisted Magnetic Recording (HAMR) has emerged as a temporary solution to the electric field limitation[34]. Through the use of surface plasmons to get around the traditional diffraction limit, it is possible to increase the bit density of HDDs to 5 terabits per square inch (tbpsi). In the future, it is likely that Heated Dot Magnetic Recording (HDMR), which combines HAMR with bit-patterned media, will be used as well. While HDDs are clearly a fading technology due to their power requirements, they can serve as a temporary solution due to the extremely low bit-cost while more robust solutions are developed.

1.5.2 Magnetic Tape

Similarly to HDDs, magnetic tape storage was a popular archival technique historically that got phased out, but due to density increases has seen a revival[35]. Given that there is no power usage after the data has been written and the extremely low cost per bit, this technology is a much cheaper storage method than many alternatives. Furthermore, due to the "air-gap" that is provided due to disconnected tape cassettes, there are major security benefits over server farms. Even more, the natural error rate on tape is 4-5 orders of magnitude lower than that of even HDDs, which already have extremely low error rates. Combining all of this with the extreme expected lifetime of tape, it is clear why this technology is making a return, although for a slightly different use case.

While HDDs and Flash memory are commonly seen as having a higher areal bit density, tape has the advantage of a much larger surface area to work off of. Furthermore, much less work has been done in terms of scaling, so Moore's Law

will continue to be valid for the foreseeable future. In contrast to HDDs, which are limited to the previously discussed superparamagnetic limit, tape has the ability to store information to a much finer detail, making use of barium ferrite particles to store information nearly 10 times as dense as traditional HDDs can[36].

A number of different advances have allowed this improvement to tape architecture. To improve the lifetime, the recording layer itself has changed a number of times leading to a complete redesign of the read and write heads. Track size scaling was also a major development, allowing significantly denser patterning than before. Future scaling suggestions include reduction of read head and media spacing and general scaling reduction along the write paths. Given that these features are much larger currently than those in HDDs, this suggests that minor improvements need to be made, not requiring major breakthroughs required to enable other technologies. Because of all this, magnetic tape is once again a clear front-runner for deep archival storage methods.

1.5.3 Other Theoretical Memory

In addition to those memory solutions there are a large number of emerging solutions that must also be investigated. At the forefront of these is domain wall memory (DWM), also commonly called racetrack memory, which makes use of defects along a material surface to store information[37]. At the core, defects are injected along a ferroelectric nano-wire using spin-polarized electric current. These have the option of being much denser than magnetic memory and can have an extremely high lifetime, making them applicable for nearly all memory types. Initial work has shown that a long retention device with a high ON-OFF ratio is possible, making this much better than many other alternatives, which suffer from very low coherence levels. This technology is far from mature however, requiring a much higher degree of material domain coherence than is possible at a die level currently in production. Due to the

ability to store more than one bit per effective cell and the 3D nature, this presents a logical solution to archival storage.

Similar to DWN, ferroelectric RAM (FeRAM) works off of single transistor with a capacitor to switch an intrinsic electric dipole to record data[38]. Unlike DWM, this is a destructive read process, and has numerous device lifetime limitations. Over time, there is a certain level of fatigue that limits the uses for DRAM alternatives, but this is still promising for an archival regime.

Magnetic RAM (MRAM), the successor to FeRAM presents itself as a better memory solution due to potentially lower read latency, higher density due to 3D stacking, and better data retention during power loss[39]. Their operational principle is based on a spin valve separating to ferromagnetic layers, where one is a pinned layer and the other is a soft layer. Read and write operations are done similar to FeRAM. Devices which operate based on Spin-Transfer Torque (STT-MRAM) has been shown to be even better, but aren't at a larger production level[40]. While this is a technology much closer to scaled production, there are issues with coherency in large systems, where each element can interfere with the surrounding cells.

One final solution is that of Phase-Change memory (PCRAM), which generally makes use of chalcogenide-based materials to switch between a crystalline and amorphous state to behave as a '1' or '0'[41]. The write latency of these devices is similar to that of MRAM and has excellent scalability, making it useful for archival storage. While also still in an early phase of development, it is suggested that power usage will decrease enough to make this feasible.

While there are still numerous other memory solutions that are possible to fill this niche, they are either too far from production to mention, or have much larger issues to sort out for a large system.

1.5.4 A Brief Comparison

With a brief explanation of these memory types completed along with a rough feasibility time-line, they can be loosely compared using a number of metrics in the table below. These values are compiled from a number of sources and are generally estimates, but serve to get a loose sense of the benefits from each technology. In particular, the realistic memory density provides a metric for what density has been provably demonstrated, rather than a theoretical limit. This becomes even more complicated in the cases of 3D memory solutions, where there is no known limit to vertical cell stacking. The power consumption metric is the expected power to perform a single bit write. In all of these cases, there are exceptions such as limited volatility, minimum write size and others.

Table 1.1: A table comparing all of the previously discussed archival memory solutions in terms of basic metrics and lifetime. All cells marked with an 'Unknown' are due to a lack of significant data.[1, 4, 5, 34, 37, 36, 40, 41, 42, 43]

Memory Type	Realistic Memory Density (cm^2)	Single-bit Write Energy	Expected Drive Lifetime (Years)	Read Time
DNA	215 PB/gram	Unknown	100+	∼10 Hours
HDDs	0.77 TB	1.5 nJ/bit	5-7	1-10 ms
Flash Memory	8.5 GB	1 nJ/bit	1-2	5-50 us
Magnetic Tape	23 GB	1 nJ/bit	10-20	1-10+ s
DWM	100-500 GB	Unknown	Unknown	1-10 ns
FeRAM	1-10 GB	5 nJ/bit	10	60 ns
MRAM	10-50 GB	3.2 nJ/bit	10	1-50 ns
PCRAM	5-20 GB	50 nJ/bit	20+	10-100 ns

This table is a good starting point for comparing the effectiveness of different archival memory types, however it is not obviously conclusive. For any practical decision to be made one must take into account all of the surrounding logic, whether it be the synthesis and microfluidic systems for DNA, or read-write architecture for DWM and others, or even the write-head equipment for HDDs. DWM seems like

an obvious choice, however there is still a significant amount of time before this technology is mature enough for a production setting. Taking all of these factors into account, DNA seems to be a strong choice purely for the much longer expected memory lifetime, but it is likely to be surpassed in the future by more novel techniques.

Chapter 2

The Current State of DNA Memory

Following a basic development of memory systems in general in addition to a development of the DNA side of DNA memory, it becomes necessary to discuss the existing works in this sector. This chapter covers an introduction to a spin-off in the form of DNA Computation and then discusses some of the major focuses in DNA memory such as encoding methodologies and error correction. It then finishes by discussing the most recent research in this area.

2.1 DNA Computing

Perhaps one of the most interesting other branches of DNA research is that of DNA computation, specifically the use of biological components to act as actual compute blocks. Originally developed to solve combinatorial problems in 1994, the proof of concept was the development of a directed Hamiltonian path problem, which is somewhat similar to the traveling salesman problem[44]. At a high level, the initial graph can be encoded into DNA strands and the computational operations are performed through various helper enzymes. In particular, this type of problem is computationally difficult because the worst case complexity is exponential. Sparing the specific details, it is possible to use an excessive amount of different DNA chains to calculated possibilities at runtime. this has problems with scaling though, not due to scaling of the paths, but instead the exponential increase from the notes to provide a de-

terministic answer. Additionally, the solutions become more probabilistic in nature, requiring confirmation that the proposed solution is correct.

DNA computation instead chose to move towards the field of binary evaluation problems due to high parallelization ability[45]. In particular, boolean circuit development can benefit greatly from an almost brute force type approach rather than relying on a costly depth solution. Given that boolean circuits can be of arbitrary depth with an unbounded number of inputs, these quickly can become difficult to evaluate, such as in modern CMOS design. Thus by encoding all of the specific situations in a similar method to the nodes presented earlier, it is possible to simultaneously evaluate operation of the design. It has been shown that this can scale up to billions of gates, which makes it reasonable for modern integrated circuit design.

While still not an active technology, there seems to be a high probability of future interest in active boolean monitoring of biological processes which normally require significant computational power. This primarily concerns the medical industry, with the potential ability to in vitro monitor combinations of molecules[46].

2.1.1 Activity Detection

One important memory parallel that developed from this was a desire to more accurately detect DNA sequences, leading to many proposed small form concepts ranging from Chemical Field Effect Transistors (CHEMFETs) to resistive measuring devices. In 2006, M. Barbaro et. al. developed one of the first CMOS style solutions for DNA memory detection, paving the way for potentially much faster and denser methods than currently in existence[47].

Up to this point, the commercial approach to the measurement of hybridization was through the immobilization of the target, followed by the sequence labeling ds strand reassociation of homologous ssDNA sequences and optical detection. While extremely robust, this is extremely slow, requiring both a chemical targeting and

expensive optical measurement tool. More modern methods are centered around cantilever designs as well as quartz crystal microbalance. The proposed solution in this work however was this to focus on a CMOS design through the use of electronic detection. Figure 2.1 depicts the proposed structure, highlighting the immobilization region as well as the general device footprint.

This device is similar to that of the floating gate flash memory technology, as seen by the extremely large floating gate in S2 (Figure 2.1). This difference in charge between the floating gate and the DNA then leads to a gate voltage and thus an encoded bit read. Unfortunately, unlike modern memory designs, this has numerous issues due to the inability to charge shift on the floating gate near the control, but techniques exist which can correct this from an architecture standpoint.

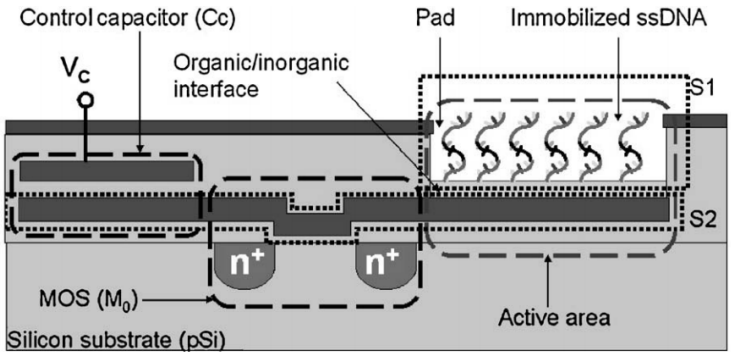

Figure 2.1: Cross-section of a Complimentary Metal-Oxide-Semiconductor (CMOS) biochip for combination detection. [47].

This design ultimately was fabricated with sensors ranging from 40x40 to 100x100 um, which while large is still significantly smaller than an optical-based approach. Additionally, this testing was done in a low strength buffer, which would be reasonable to use in a full micro-fluidic system. During fabrication, a surface activation procedure was used, which does raise concerns though about the durability of this design, particularly in the high demands of memory technology.

While this is not true memory decoding and rather simply a method to detect the combination of a DNA sequence, it could potentially play a role in future microsystem

designs. Future applications of this work could be in write verification, such that the stored strand blocks are systematically tested to ensure that they are roughly the correct length, suggesting that they are bonded correctly.

2.2 Encoding Methods in DNA Memory

Moving back into the memory side of applications, it quickly became necessary to develop more rigorous encoding methods, particularly to minimize the number of naturally occurring errors. While the obvious solution of a direct conversion of binary to base four is possible, it leads to an unattractive number of synthesis errors due to the strand randomness[48]. At the forefront of solutions was the concept of using Huffman encoding to translate ASCII characters based on their statistical likelihood and converting them into known low error base sets[49].

2.2.1 Huffman Encoding

Originally proposed in a work from Smith et al. in 2003, Huffman encoding quickly became the de facto representation of how DNA would be encoded to efficiently store data. Using a probability distribution of characters, each character is assigned an individual set of base pairs to act as the code. An table using the probability distribution of letters in the English language is shown below[50]. This allows for the easy construction of a sequence with fewer than randomly suggested errors. Furthermore, by designing the encoding around the data source, it is possible to achieve an average of 2.5 codons per character encrypted.

Table 2.1: Example of encoding based on character probability in the English language.

Character	Probability	Sequence	Character	Probability	Sequence
e	12.7	C	w	2.4	GGC
t	9.1	GA	m	2.4	GTG
a	8.2	GC	f	2.2	GTA
o	7.5	AG	y	2	GTT
i	7	AA	g	2	GTC
n	6.7	AT	p	1.9	TTG
s	6.3	AC	b	1.5	TTA
h	6.1	TG	v	1	TTC
r	6	TA	k	0.8	TTTG
d	4.3	TC	j	0.2	TTTA
l	4	GGG	x	0.2	TTTT
c	2.8	GGA	q	0.1	TTTCG
u	2.8	GGT	z	0.1	TTTCA

Other works added to this by either adapting it for image storage, or audio recordings[51]. While this is by no means a universal encoding method, is has numerous benefits in the case of direct text recording, although it would be extremely simple to extend this for numeric encoding.

2.2.2 Other Proposed Methodologies

Further alternatives to the proposed Huffman code were investigated by many of the same authors, but they are focused primarily on pure text encoding, which is far from universal. In particular, they proposed two important new encoding algorithms, comma code and alternating code. While individually not that useful, they provide a basis for more modern works as well as highlight a significant problem. As this is an introductory work, they ignored implications of error protection by naively making

the statement that replicate chains can be used, which destroys the efficiency.

In the case of the comma code, the conceptual idea is to use five base codons, which are separated by a single base which is denoted as the comma. The remaining slots are then filled with the remaining base pairs which make up the character sequence, thus providing 3 pairs, comprised of the two logically opposite bases with the comma invert as the third. This provides a logical reading frame, which provides protection against insertion and deletion mutation, for little special cost.

The other proposed method of encoding is that of the alternating encoding pattern, which presents a method of 6-base codons of alternating purines and pyrimidines. Similar to the comma code, characters are set in a 6-base pattern, but in this case have no frame protection. This has the advantage of being able to establish exactly where the error occurred, but is largely space inefficient.

These coding methods are less efficient than the Huffman encoding, but have unique advantages from secondary points of view. They do contain significant costs in terms of computational complexity, which is going to become a significant, particularly when it is stacked on top of other data protection techniques. It becomes important to find the file-size at which it is more efficient from a write latency point of view, encoding errors notwithstanding. With the realization that all written strands should fall within a range of a few hundred to a few thousand base pairs, this becomes an important conundrum. While a much longer strand length would be desirable due to the naturally higher density, the shorter strands make it much simpler to attempt a read correction.

More importantly, a key focus suggests that there should be some identifier to detect if a DNA sample is a data source or of natural origin. In the event of contamination, it might become necessary to disentangle usable data from organic contaminants.

A lot of these works have delved into an improvement upon the original Huffman

encoding techniques, but there seems to be a focus on character efficiency rather than looking at a fully system level. In particular, while the individual characters are important, the system overheads end up being significantly more problematic in terms of space efficiency as well as pool recall time (PRT). Notwithstanding, Ailenberg et al [51] were some of the first to propose a method better than the original Huffman encoding that is more universal than text specific.

This work chose to define a DNA base segment to represent a character, for all of the keys on the keyboard, which can then be simply inserted as a base pair sequence. Compared to the previously described methods of comma or alternating codes, which provide a ratio of 6 bases per character, or sequential encryption, at 5.3 bases per character, this approach chose to minimize the character set to a basis of 26 characters, where secondary characters are referenced by specific longer chains. Extending upon this, it becomes possible to efficiently encode other file types in a similar fashion, where the number of operations is reduced to a trivial set of instructions, which can then be assigned to valuable sequences. Simple graphical images can similarly be encoded by defining vectors in the image sequence, and then using a decode library to interpret the results. It was shown that using these methods, text-based encodings could be done at 3.5 bases per character with other tested sets at 4.9 bases per character. In specific cases, this shows its value but runs into the issue of requiring prior knowledge about the file itself, making it non-universal.

While this is a valuable work, it complicates the issue of encoding by making the sequencing much harder. Rather than running the system through a cypher to generate the encoding, there is now an overhead of translation libraries, which either have to be statistically generated, or hard-coded. From a system perspective, the memory controller chip has to either assign its own codes dynamically, which becomes problematic for an encoding perspective, or instead have the front processing system provide encoding suggestions, which is unreasonable. Some would argue this is trivial,

but for smaller data packet sizes this hurts the system response.

Furthermore, Goldman et al.[52] presented what has become the default suggestion of DNA encoding through a modified Huffman encoding. The sequencing process is shown below in Figure 2.2.

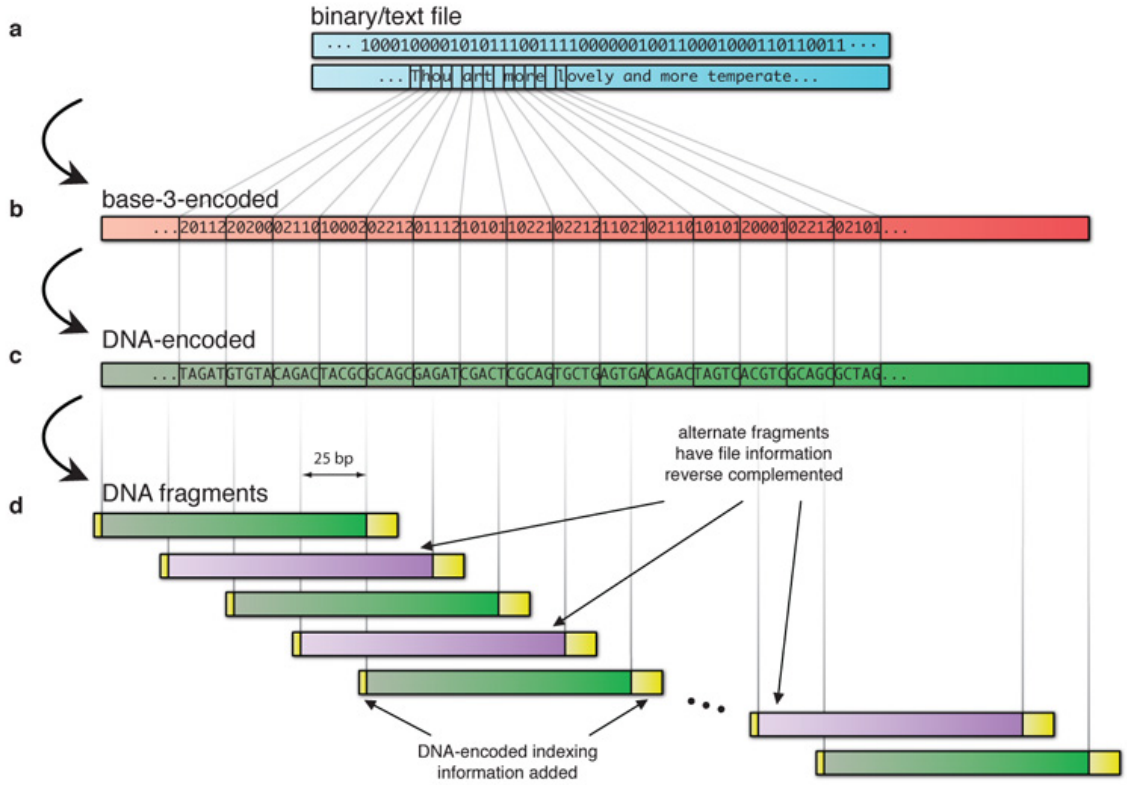

Figure 2.2: Depiction of full-pattern encoding sequence for ssDNA showing translation into the base three system and finally the DNA encoding[52].

This strategy combines a number of previously developed encoding methods into an operation function, beginning with a Huffman encoding into the necessary base three system, converting local regions into stable DNA codes, and separating into specific small-length strands. This allows for both a dense storage regime in addition to an adjustment region to eliminate short-chain errors. Furthermore, all tagging information can be supplied at the beginning of encoding and strand caps synthesized at every end.

One important concept to note from this work is the use of a next state table

to determine the next base in the sequence. This is one of the most important metrics to limit the number of repeated bases in a sequence. A table such as the one below could be used to prevent this repetition. Furthermore, deeper state transition diagrams could be used to avoid many other short chain risks, with the decode logic simply reversing this process.

Table 2.2: Next-State diagram demonstrating a method to prevent prevent homopolymer runs of any length.

Current Base	0	1	2
A	T	G	C
T	G	C	A
G	C	A	T
C	A	T	G

2.3 Error Correction Methods for Modern Memory

Given the previously explored high error rates of this design, it is next important to develop a thorough background of both what error correction algorithms and what usefulness they have in the field of DNA memory. In particular, Reed-Solomon codes with their successor Bose-Chaudhuri-Hocquenghem (BCH) codes are highlighted to show what natural synthesis error can be corrected. Additionally, error recovery methods for DNA are mentioned in the form of multiple copies of the same strand.

2.3.1 Reed-Solomon Codes

Given that error correction is so important to the future of DNA memory, a more thorough explanation of basic error correction, as well as Reed-Solomon methods are developed here. In essence, the purpose of error correction codes are a method to discover the existence of errors and correct them to recover data. Whether these errors are caused during synthesis or during the read-back process is relatively unimportant,

just that they exist. One simple example of error checking is through the introduction of parity bits, which are an extra bit added along to the end of a sequence, which provides information about the information in the chain. One simple example is given in Figure 2.3.

Data Sequence	Parity Bit
10100011101011011	1
00100011101011011	**1**

Figure 2.3: Example demonstrating the use of a single parity bit in a sequence, with the correct sequence on top and the incorrect sequence on bottom. Because the parity bit is a '1', it suggests that a bit flip has occurred.

While this is an extremely simple example, it demonstrates how a misread can be identified in a sequence. This can then be followed up by a re-read in most cases, but with DNA memory this becomes slightly harder. Because a direct read would be destructive, the read either has to be properly done the first time, with any errors being handled by the microcontroller error flow, or has to be made on numerous copies, which is a time-intensive process.

In the cases where reads can't be retried in a method similar to modern processes, it is important to have an in place correction such as Reed-Solomon correction[53]. This systematic coding scheme, which operates by adding a code word at the end of a data set has the ability to correct for multiple bits in a sequence. For some set of data with length 'k' and 's' bits per symbol, a set of parity symbols '2t' is added such that $k + 2t = n$, the total length of the data segment. From this, 't' total characters are able to be recovered. A common example of this is through RS(255,223) with 8-bit symbols representing bytes as shown below. It is clear that errors in a total of 16 bytes can be corrected for.

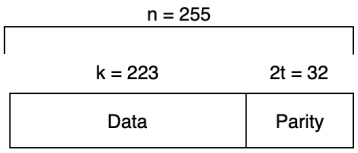

Figure 2.4: Example showing RS codewords, with the 223 B data on the left and the 32 B correction on the right.

This process of correcting errors is based on the concept of finite fields, where a Galois field is used to generate a polynomial to provide an expected data result. In a simpler sense, it is easiest to consider the idea of adding specific characters to chains to make them unique, similar to a dictionary. These extended words are then compared on the decode side to the set of possible words. Thus, any words that are different than expected, which would occur in an error, can be corrected to the nearest correct word, given by the Hamming distance. A simple example is given below using text.

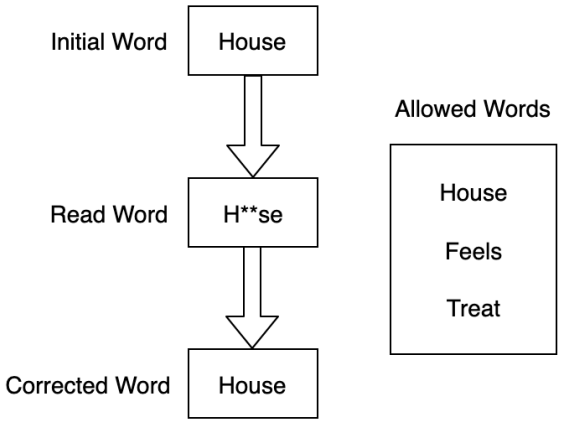

Figure 2.5: Example showing and RS text example. The initial encoded word is 'House', which is then misread as 'H**se', with 2 error values. This is then compared to the allowed word bank, where the closes word is the original 'House'. This demonstrates the ability to correct for a total of two character errors.

However, as 't' grows, increasing the ability for correct bit flips, the processing power increases significantly. While originally intended for byte codes, this can naturally be extended to handle DNA encoding, simply by handling the encoding before

conversion into the base equivalence.

2.3.1.1 Bose-Chaudhuri-Hocquenghem (BCH) Codes

While Reed-Solomon correction is extremely powerful, most modern memory error correction schemes use BCH coding methods[54]. While these two methods are extremely similar, BCH codes are a generalization of RS but operate with a lower number of codewords, making an exhaustive correction search feasible for memory read operations. BCH codes are much more attractive due to their emphasis on the correction of random errors, rather than burst errors. In flash memory for example, it is more likely for a number of errors to occur in a row due to cases of physical defects or write level issues. It is due to this trade-off of burst correction or random error correction that makes memory correction more difficult due to the occurrence of both. However, for DNA memory, single-base mutations and misidentifies are far more likely to occur than a chain error sequence, making BCH more effective.

2.3.2 Error Recovery Methods

In addition to direct in-line error correction, there are a number of different methods of error correction based on multiple chain reconstruction. The premise of this is to simply duplicate that data a number of times to ensure a correct read, similar to how modern DNA sequencing relies on numerous samples. While this is acceptable for biological applications, in order to achieve the highest data density, the goal should be to minimize chain replication. While early works were content with the statement that four or eight copies of the data were sufficient, it was clear that better methods existed[55].

One such method was that of data overlap, where different strands would contain shifted copies of the same information, thus reducing the likelihood of repeated errors at the same position. Bornholt et al. improved upon this further by making use of

a three strand replication[56]. This takes the ideas from a RAID 5 setup, where two strands, A and B, are XORed together to form a third strand AB. Thus, if any two of these strands are correctly recovered, even with ECC, the last strand can be correctly extracted.

2.4 A Comparison of Existing Systems

Now that a development of the concepts behind DNA memory have been completed, it becomes important to discuss some of the full system designs that have been published over the last few years.

2.4.1 A DNA-Based Archival Storage System

Perhaps the first major work following the 2014 paper by Goldman et al. was the previously mentioned work by Bornholdt et al[56], which introduced a key-based method of storing and retrieving data. The architecture design proposed was similar to many other works, comprising of a synthesizer module, a storage container, and a sequencer module. this proposed system made use of DNA strands 100-200 bp long, correlating to 5-100 bits of data. Using a Huffman encoding to convert the data into a base three system, a rotating encoding system was used to translate all of the data into the DNA nucleotides. The data itself was stored in strands as a payload combined with an address surrounded by a sense nucleotide and strand primers, which are used to assist in the read portion.

In particular, careful attention is taken to the concept of key pair applications for DNA storage. Due to the fact that the DNA itself has been proposed to be stored in microfluidic chambers, it is assumed that the controller itself will know where the data is physically stored. While this paper didn't explore the details of this, it seems likely that a simple address translation can be used to provide a pseudo-location front the front end perspective, while the system microcontroller can rely on internal flash

tables or similar to handle address decode. Thus, a conundrum is reached on how to ensure random access. If pools were kept small, thus increasing DNA separation and reducing the minimum read size, this vastly reduces the total spacial efficiency. However, at the opposite end, if pools were kept large, minimum read sizes are greatly increased, leading to much slower access times. This work discussed a potential fix to this issue through the use of mapping keys to PCR primers. While the identification of specific primers was outside of the scope of their work, it is an extremely valuable discussion of how to minimize spacial inefficiencies. This allows specific amplification of specific strands at read time, increasing the likelihood of a correct read in a specific pool. Limitations on pool size are then simply based on the number of unique effective primer agents.

A total of four different files were successfully encoded and decoded using this process, ranging from 5kB to 84kB, showing that it is possible to write files of a size reasonable for a single storage pool. While this work is somewhat old, it was important in presenting a method for a full system design of DNA memory. One key highlight is that even with these small file sizes, they did encounter errors, which were manually fixed with the understanding that future work needs to go into error correction using BCH or a similar process.

2.4.2 DNA Fountain

The next major work to discuss is that of the DNA Fountain, which is an early demonstration of the ability to store larger computer files in DNA memory and successfully decode them[57]. This work provided a better analysis of the true Shannon information density, which was found to be 1.83 bits/bp, rather than the ideal 2 bits/bp proposed before. This work combines some of the encoding methods previously discussed with the use of RS for error correction to reduce the level of DNA overlap for data duplication.

The encoding process is somewhat different than the other explored ideas, where data is first broken up into non-overlapping chains of some decided length. Next, the data is iterated using two distinct steps, a Luby transform and a screening. The Luby transform is responsible for collecting data into a set of packages of short sequences, called droplets through random selection and bitwise edition under a set field. Each of these droplets contains a payload and a fixed-length seed, which provides a state of the random generation which chose the data distribution. The screening step of the process then translates the specific binary sequence into base pairs directly (00 to A, 01 to G, ...) rather than a shifting cipher. The chain is then screened for the content of GC and homopolymer runs, which would make the chain unstable. If the screen is passed, the data can be stored, but if it fails a different encoding seed is chosen and the process is repeated.

From this method, a total of $2.14\text{x}10^{6}$ bytes of data were successfully encoded and decoded, requiring nearly nine minutes of scripting time to verify. This was then shown to extend to a maximum of $2.18\text{x}10^{15}$ retrievals. Unfortunately, this method is computationally costly, which makes it impractical for a full-scale system. Potentially future work will find a way to reduce the computational complexity, or instead find a way to create dedicated hardware to speed up this encoding.

2.4.3 End-to-End Automation

2019 marked a transition into larger examples of fully automated end-to-end systems, which marks a large step forward. While the vast majority of works up until this point required physical interaction as some stage of the process, Takahashi and coworkers developed a fully automated system to handle DNA memory storage[58]. Even through it is unable to store nearly the information that some of the previous works have shown is possible, it provides a scalable proof that a truly automatic system is possible.

Using similar methodology to previous works by including error correction and adapters, encoding was done on chains longer than 1000 bp, which is significantly longer than other works, which typically have aimed for 100-250 bp in length. The overall system latency was seen to be 21 hours, with the vast majority of time being spent on synthesis at a rate of nearly 305 seconds per base. This aligns with other works, which suggest that synthesis will always be the major delay in design. In contrast, the actual read portion of the synthesis took less than an hour, showing that reads can be done in a somewhat reasonable amount of time.

A 5 B message was successfully encoded and decoded automatically, showing successful operation. Unfortunately, this setup was extremely poor in terms of overall efficiency. Out of the 1 mg of DNA synthesized to store this information, only 3469 reads were sequenced, of which 1973 were aligned with the specific adapter sequence. Furthermore, of these only 30 had extractable payloads, with only a single correct read. All other read attempts were identified as unrecoverable, but this wasn't elaborated on. This shows that while other works have shown that this informational encoding is possible, there is a large gap between what is possible and what can be currently automated.

2.5 A Highlight of DNA Security

One final important sector to note is that of the security of DNA. Unlike other forms of memory, where it is impossible to physically read out of system, gaining physical access to the media allows for direct reads in the same manner that the DNA would naturally be decoded. Thus, it becomes important to ensure a robust security method to limit recoverable information.

While a number of solutions exist for other memory types, recent work has focused on the concept of a physical key decode, where the DNA is only able to be properly decoded on a specific system, as explored by Chen et al[59]. Due to the unique manner

that DNA is read, there are advantages such as the use of these physical decode keys, which will have only a minor effect on sequencing times. By introducing additional single-sided overhangs on a side of the double helix to act as security key. Thus, in order to correctly decode the DNA itself, the strand has to be read into the correct micropore which contains the proper complimentary strand key leading to correct data decoding. If reads occurred at any other location with a different key, the data would be misidentified.

It is noted that this is an imperfect solution, particularly because DNA strands such as those suggested will have a significantly lower level of stability, which Chen et al. estimate will only survive for 30-32 years. In the long term, it seems likely that a different method of physical encryption will be necessary in order to ensure the 100+ year lifetime targets. One likely decode solution for the decode method would be to directly encrypt along the chain, where the nanopore would pick up the matching address decode key. This could be handled completely in system, where the encode logic knows the physical location of the data storage and because of that, it would know which nanopore would be responsible for that specific read at decode time. Read error recovery could then potentially reroute DNA decode into other similar pores based on some other internal architecture. The physical location would be translated from the system logical address, thus preserving locational integrity similar to modern systems. While this falls prey to a strict dependence on the nanopore integrity, the relatively large size would make it less prone to failures. This idea hasn't been properly explored, but would be worth investigating.

Chapter 3

Defining DNA Memory Architectural Systems

Following the development of DNA memory systems, it is natural to compare existing works to determine the most effective methods of design. Some of the key recent works are identified, they are slightly modified to provide equal grounds for comparison. The proposed method of deep state-based encoding is then developed using background theory. In particular, many works gloss over the concept of an ideal chain length, choosing a length based on their synthesis and decode tools and adjusting the work to fit the system. Instead, a memory-oriented base length should be developed as a target.

After determining the ideal chain length size for a hypothetical system, it is then important to develop a larger memory block by defining pool sizes, data distribution, and microfluidic array size. Given how few physical constraints exist due to lack of published information, a large number of assumptions have to be made. Special care is taken to highlight a proposed interface between the DNA pool readout and the physically proposed hardware, given that this will be extremely computationally complex.

Finally, a proposed system is developed to provide a hardware flow on the input and output side of the theoretical drive. While this in no way a complete end-to-end design, it provides an initial hardware implementation of how such a system could be constructed in hardware rather than software.

3.1 Encoding Comparisons for DNA

Unfortunately, given the low number of different works in this sector, there are often discrepancies between papers as to the exact metrics used for comparison. This section develops the metrics needed for any encoding strategy in terms of net bit/bp as well as a total chain efficiency.

The first primary metric is the coding potential of the data, which represents the ideal number of bit/bp possible following the rules of the encoding pattern. This value is capped at a theoretical maximum of 2 bit/bp by assuming absolutely unique bp targets with no automatic redundancy. In the future, if sensing techniques continue to improve, there is a possibility of increasing this value further based on different bond-lengths such as the AT versus A*T*, which differ based on the hydrogen bonds between the two ssDNA sides, and treating those as unique. Presently, this is unrealistic, and deemed outside of the scope.

That metric is a strong baseline, however it is a precursor to the net information density, which is a much harder metric to meaningfully quantify. While the calculation itself is quite simple, simply being total number of bits / total number of base pairs, this provides a bias for systems which minimize redundancy and assume perfect storage location information. While that is closer to ideal in terms of pure storage density, full systems will forgo that to provide information on addressing, error correction, and in the case of DNA memory, decode handles to improve read times.

Another similar metric commonly used is the net redundancy of information, which is the amount of straight data repetition. In early works, this was characterized by a high value due to chain duplication, however more recent works have lowered this value closer to the target of 1, where there is no direct data duplication.

The final two metrics to discuss aren't specific to DNA memory, but instead tar-

geted towards the proposed memory architecture. In particular, the total addressable space is important because it has to handle the high theoretical memory densities. Given that this is a metric that hasn't been the target of much work in this sector, there are vast improvements possible through design tweaks. Furthermore, the minimum memory block size needs to be identified because it puts a limit on the minimum write size to the system. This value is extremely hard to compare because most works weren't designed to be put into a full-scale memory environment.

3.2 State-based Encoding Methodology

While the Fountain approach of encoding has been shown to be vastly more efficient than a simple next-state encoding, it comes at an extremely high computational cost. While the naive thought would be that due to the extreme encoding times, this is trivial, it unfortunately is not given the scaling needs of write execution. For this technology to be feasible, there needs to be a massive parallelization of encoding, rather than the linear encoding methods used in prior works. Thus, the target should be a modular encoding pattern that can easily be implemented in hardware to develop encoding patterns for micro-array synthesis. Thus, a simple encoding method must be developed which requires relatively little hardware logic and achieves similar encoding density to modern work.

In development of such a pattern, two main issue arise which need to be accounted for. There exist issues with underrepresentation of read coverage in regions of high GC or high AT content, which suggests that data should be kept to an even distribution locally as well as globally[60]. For this work, the encoding target was between 0.48 and 0.52 GC bias. One oversight of other works is that they target the entire dataset as a whole, when in reality the address tags and the payload itself can be treated separately. Separation of these regions is less of an issue due to the large number of amplifications that occur. The second important requirement is the limitation of

homopolymer runs as discussed before. While some works strictly limited polymer runs to one in length, it is seen that limiting below a total of four is reasonable[61].

3.2.1 Ideal Chain Length

One important portion of theory that is important to discuss is the dropout rate during PCR amplification on the read-side of the system. In particular, in larger pools, it becomes important to ensure that the specific read chain is being amplified entirely, which is based off a probability mass function, shown below.

$$\delta_v = \left(\frac{\mu}{\mu + r}\right)^r \tag{3.1}$$

In this equation, r represents a process given factor determined by the library processing technology, from 2-7. This value is roughly inverse to the amount of PCR rounds that the sample undergoes. μ is correlated to the sequencer capacity, which when taken to be a reasonable value demonstrates that a dropout rate of 0.5% is reasonable.

However, with specific handle targeting for replication, this can be reduced low enough that this is unimportant. This is identified though to demonstrate the importance of PCR targeting. Using a common metric, it is possible to design for primers 18-24 bp long, which lead to a limitation of 150-1000 bps of remaining chain length[62]. Using the RUCS program, following the rules of the following section, it is suggested that there are 800 potential 22 bp primers to work with in this case[63]. This is much smaller than the expected addressing due to the requirements around repeats, GC content, and other factors in the primer itself. By treating each of these as a unique address key, it is possible to then treat each strand as an addressable location, rather than having to decode the entire pool. Logically, a secondary address conversion table can be used to convert the front end address label into the pool address and PCR key, which will be discussed in the following section.

Optimization around maximizing PCR throughput with a limit of 996 bps, working around the modulo six requirement from the Raptor analysis, defining the minimum read size[64]. While primers can be developed for longer than 3-5 kbps, they require additional time and provide further restrictions on the design. Using this, after subtracting the two primers, this places a theoretical upper limit of 764.8 kbps for a specific pool, with individually addressable memory of 956 bps segments. This efficiency however is subject to the limitations of BCH encoding and the state-efficiency loss as stated above. A depiction of this strand/pool arrangement is shown below.

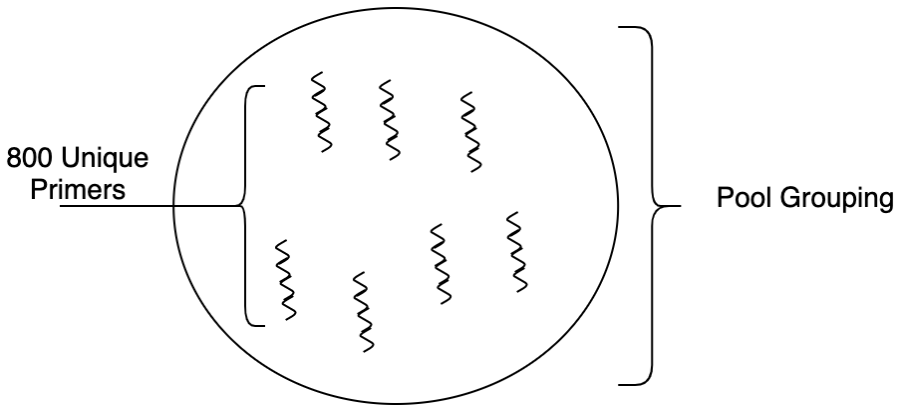

Figure 3.1: Example showing an individual pool with DNA strands. In particular, this provides safety for under-sampled groups.

3.2.2 Use of a Raptor Encoding Pattern

Given the vast improvement in efficiency seen by the Luby transform (LT) to increasing coding efficiency, it is natural to lean in this direction, provided a more efficient encoding and decoding method is found[65]. Luckily, this exists through Raptor codes, which are an extension of LT codes but linear in encoding and decoding, making them much more easily implemented in hardware[66]. Thus, the original condition from LT that all input symbols must be decoded, the decode graph can be constructed out of a number of edges equal to the number of input symbols, making it linear. Instead, if the natural read and write errors are minimized, Error Correction Codes (ECC) can

be leveraged to fill in the remaining bits, thus increasing informational density.

By considering the dsDNA strands as message packets, it is easy to demonstrate how this scheme can be applied to DNA memory. This code is really a modification of the original LT fountain coding, being comprised of a total of two separate codes. However, unlike LT, pre-code-only (PCO) Raptor codes use the simplest possible output code[67]. To develop this, an R10 Raptor code in a fixed rate setting was adopted for use in this work[68]. In particular, each pool is treated as a single codeword for the Raptor algorithm. The relative position, in a memory sense rather than a physical sense determines the key used. The foundation of this is based off of the presented Fountain scheme, where a source of K symbols is used to produce a set of output symbols M through a linear combination, where $K = M$. These packets are generated to match the number of specific strands in a pool. A flow of the encoding pattern is shown below in the form of an example.

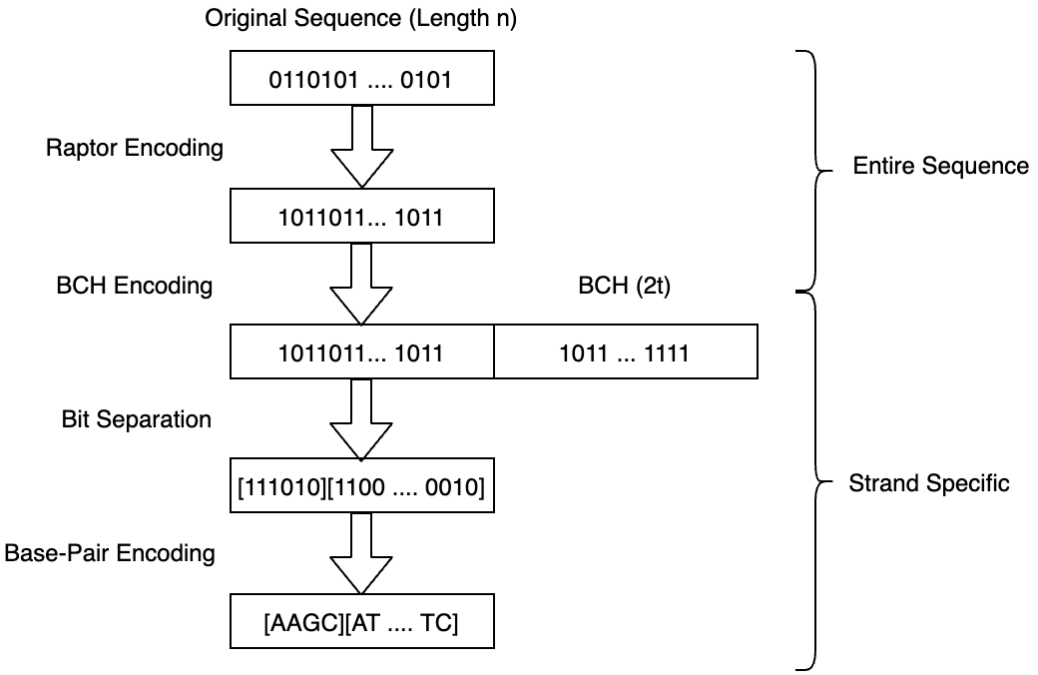

Figure 3.2: Example showing the flow of data encoding. It is important to highlight that this is the data specifically, which doesn't include the specific key.

Thus, the entire system can be defined as an input function $f(K, C, t)$, where K is the set of input symbols, C is the encoding scheme, and t is the number of correctable bits required. By limiting the redundancy overhead, the time complexity is reduced to $O(K)$ rather than $O(KlogK)$ at the cost of a higher error rate.

Developing the ideal of the Raptor combined with BCH, the Raptor code is applied to the entire pool of data, to form an extremely large codeword, comprised of a combination of symbols taken from words in various individual strands. In this context *word* denotes an arbitrary length n. The encoding process is done through two parts, beginning with the generation of intermediate symbols. Defining **m** as a vector of L intermediate symbols, along with $t = [z^T s^T]^T$, where z is a zero vector of length (S + H), which itself represents the relationships between intermediate symbols, and s, which is a vector of k source symbols, and encoding relationship can be defined. A_{pre} is an LxL matrix of GF(2) (Galois Field).

$$A_{pre}m = t \tag{3.2}$$

In this case, $L = K + S + H$, giving a total array to provide a binary relationship between all of the initial symbols and the intermediate symbols. This can then be easily adjusted to solve for $m = A_{pre}^{-1}t$.

The second step involves the generation of all parity symbols, $(N - K)$, which is calculated from the G_{LT} of dimension (N-K)xL binary matrix. This can be used to generate individual parity symbols r.

$$r = G_{LT}m \tag{3.3}$$

Combining these, the overall coded vector can be calculated as follows, where step

one can be preprocessed leading to a non-trivial, but linear solution.

$$C = Am, A = \left[\frac{A_{pre}}{G_{LT}}\right] \tag{3.4}$$

A is then known to both the encoder and decoder. The decoding can then relatively simply be done through Gaussian elimination on $A'm = c'$, where A' is the reduced generator from A, due to dropped symbols.

The likelihood of an uncorrectable error of the inner-code can then be computed using the following, where p_e is the raw bit error rate, which can be loosely calculated using the synthesis and sequencing error rates.

$$P_E = 1 - \sum_{i \leq t} \binom{n}{i} p_e^i (1 - p_e)^{n-i} \tag{3.5}$$

The overall probability of an overall uncorrectable failure can then be found as follows. It is assumed that the undetected error probability of the inner error correction, is orders of magnitude below that of the other errors, and thus can be ignored[69, 70].

$$P_F \leq \sum_{i=1}^{i=k} \left[\binom{K}{i} P_E^i (1 - P_E)^{k-i} min\left\{1, 2^{-(N_s(N-K)-N_s i)}\right\} \right] \tag{3.6}$$

Following this, the final translation is done by a mapping of data bits into using a scheme that maximizes efficiency while maintaining the strict biological limitations of DNA. Using this development, combined with the physical pool sizing requirements, it is possible to design a system to maximize the data throughput while still maintaining an error rate similar to what modern memory can provide.

3.3 Defining a DNA System

Using this encoding scheme, it next becomes important to develop an outline of a conversion layer for the memory system itself. While it is clear that individual strands

can be made addressable due to PCR, the ability to read a single strand is nearly pointless due to the encoding pattern used. Instead, the front end memory only needs to keep track of the pool as a whole. The individual addressability can then be treated as an error recovery mechanism, where a sequencing mis-identity can be corrected through a specific re-read rather than a complete re-read. While not the focus of this work, it is important to highlight as an advanced recovery mechanism. A high-level diagram of the complete system is shown below.

Due to the complexity of the design, only some of the key portions were developed in VHDL 2008. This included the address conversion layer, an example microfluidic communication layer, and the encoding module. On the output side, a theoretical microfluidic adapter is proposed as well as error recovery logic. The address itself then needs to be transformed into all of the required components. Ignoring the obvious logical to physical address conversion that occurs in most memory systems already, the address in a DNA system has a much larger functionality. In addition to providing a physical address, this also is used as the key for the Raptor encoding and defines the adapters used as well. A flow of the encoding side of the design is shown below.

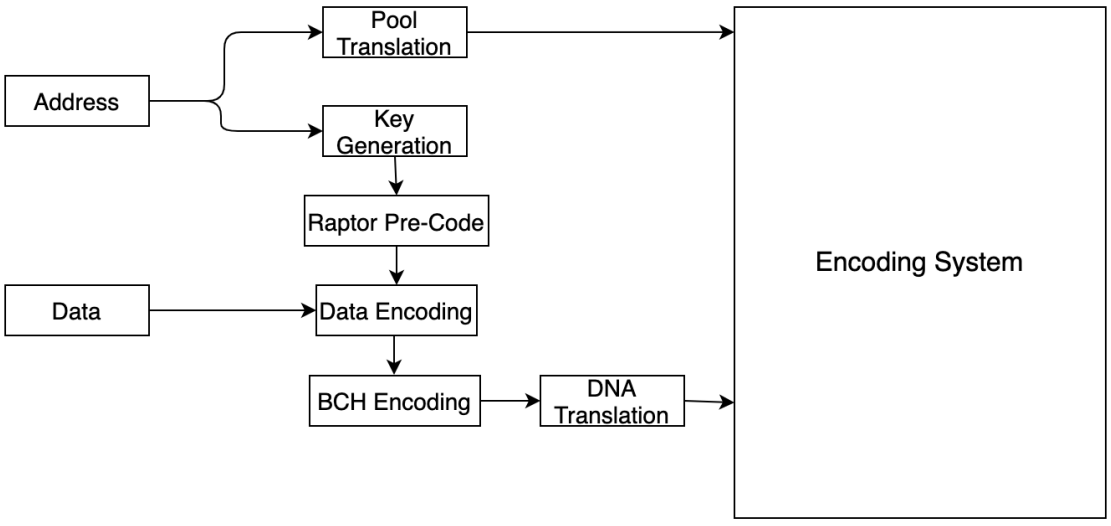

Figure 3.3: Front end of encoding system, demonstrating full flow from write to store. This is treated as independent to a full memory setup to limit overhead.

In particular, the design of such a system implies that the number of sequencing sites *ss* is orders of magnitude lower than the number of storage pools total, leading to a necessary microfluidic movement. With the extreme length of sequencing time required compared to traditional memory, read requests are then highly dependent on the ability to make read requests, then arbitrarily move samples between read sites so that performance hits aren't as dependent on read location. A high-level design of this decode architecture is presented below.

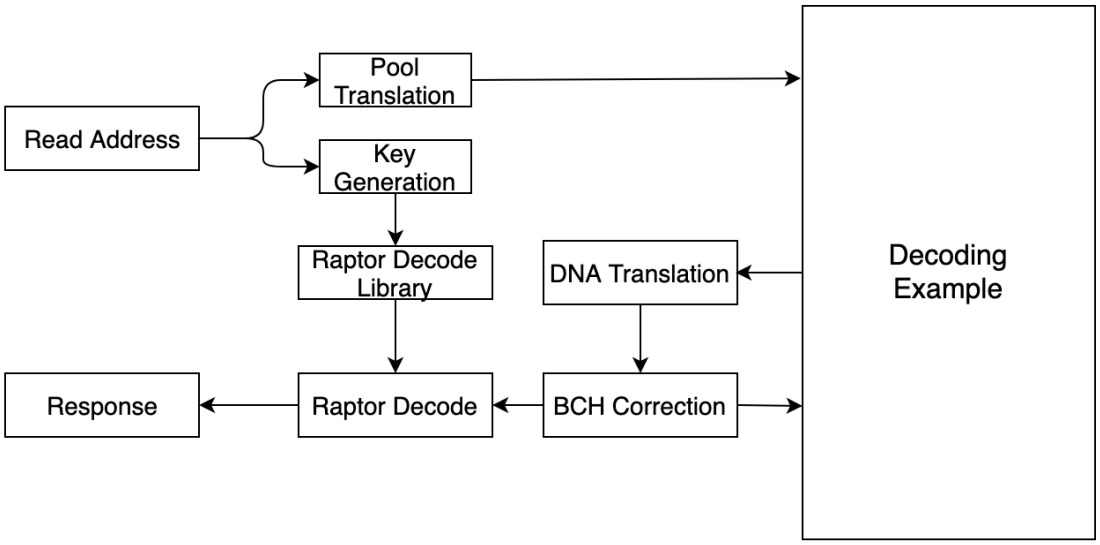

Figure 3.4: Decoding side of the DNA storage system. The actual design is much more complex due to the more complicated logic in the region of DNA translation to the BCH correction in the event of a read-retry.

Using these encode and decode units, it is possible to approximate a system by simply tweaking the physical biological unit parameters and making adjustments to the adapter modules.

3.4 Evaluation of DNA Architectures

Taking this design with the calculated parameters, it is then possibly to provide rough throughput measurements, particularly on the decoding side of the applications. Given that the encoding is always going to be limited by the ability for individual

base synthesis, the ability to access the archived memory becomes the key throughput measurement. The final portion of this work considers the evaluation of decoding throughput, under a range of sequencing assumptions. While not mathematically rigorous, this is used to suggest architecture design for a more efficient system.

In particular, given that the entire biological memory block is working off of limitations in sequencing and synthesis technology, a modular investigation of the impact of strand length on encoding performance and memory requirements is performed. Key interface requirements are also outlined to maximize encoding throughput.

Furthermore, care is taken to investigate the so-called worst case requirements under the condition of a repeated read-retry. Under a system holdup due to a BCH failure, suggesting that a PCR-enabled reread is required, it becomes important for drive recovery as well as minimal system slowdown. In particular, the decode request stack needs to remain agile enough to attempt a secondary read at any given time. Various levels of read failures are discussed as well as how a larger system should handle them.

Chapter 4

A Comparison of Encoding Patterns

Following the outline of requirements for the Raptor with BCH encoding pattern, the implementation and results are presented below. The structure of this chapter is the reverse of the encoding method, where the binary to base pair translation is given, followed by the generation of the ideal strand length based off of the BCH correction probability, which gives rise to the original pool size proposal.

An error analysis is then done showing the expected error rates between different modifications of both the BCH and Raptor algorithm as well as how this can be tuned based on specific encoding and decoding tools. The final section is then devoted to a comparison between this technique and previous techniques in terms of various efficiency metrics, along with the identification of key areas of improvement.

4.1 Direct Conversion of Binary into DNA Bases

The first important translation is that of binary to base pair. Once the entire Raptor chain with BCH is encoded, the resulting binary sequence is then converted into the base pair equivalent. For simplicity, the four base pairs are assigned a 2-bit binary code, which would then be provided to the sequencer as shown below.

Table 4.1: A table showing the binary value for each base.

Base	Binary
A	00
T	01
G	10
C	11

The conversion itself is done through a mapping of 8-bit segments onto 6-bases. While the 8-bit was chosen to match the unit of a byte of memory, attempting some reasonable level of overlap, the 6-base requirement was chosen as the smallest number of bases that could encode 8-bits after illegal segments were removed, while still leaving enough to generate unique primers for PCR.

For the case of 5-bases, a brief mathematical argument is presented below. A total of 1024 possibilities exist, which, after removing the 64 cases caused directly by runs of four or five homopolymers is reduced to 960. However, this still leave the possibility of a run of 4 or more segments being generated by a sequence of symbols. To encode the 256 possibilities from the byte of data, they must be mapped in a way that none of these runs are generated. One trivial method of generating these is to allow character positions one, three, and four to be any possibility, with positions two and five such that no run of three is created, thus providing 576 possibilities, which is more than enough for the required encodings. Given that all of the specific runs of four out of five homopolymers were removed, this proves that the entire sequence could have a GC imbalance of maximum 60/40, which was shown to be reasonable. However, even with this mapping, there aren't nearly enough to generate a unique-enough primer set to target a specific strand, thus 6-bases are required.

Taking this 8-bit into 6-base mapping leads to a total of 4096 possibilities, where after subtracting the 172 chains of 4 runs or longer, the 256 specific targets could be

mapped into a unique set. The decision was made to group these closely together, thus allowing for primers to have a maximum separation from the set itself. This set was chosen as follows, where c_1, c_2, c_3, c_4 are distinct bases and the addition indicates concatenation.

$$c_i = [c_1] + \begin{bmatrix} c_2 \\ c_3 \\ c_4 \end{bmatrix} + \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} + \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} + [c_1] + \begin{bmatrix} c_2 \\ c_3 \\ c_4 \end{bmatrix} \tag{4.1}$$

This set of possibilities was then generated in a Python script and showed that the natural GC to AT balance remained at 50%, so combinations were removed at random to provide a final balance of 50%, satisfying both key requirements. A portion of the resulting code is shown below.

Table 4.2: A table showing the first four allowed base conversions.

Binary	Base Conversion	Binary Valuation of Bases
0b00000000	ATAAAG	0b000100000010
0b00000001	ATAGAA	0b000100100000
0b00000010	ATAGAT	0b000100100001
0b00000011	ATAGAG	0b000100100010

A simple example of a conversion with some of the data is shown below, while the full table of the encoding set is given in Appendix A. A total of 4 bytes of data are encoded, broken into 1 byte segments. This binary equivalent is then interpreted by the synthesizer and turned into the strands themselves.

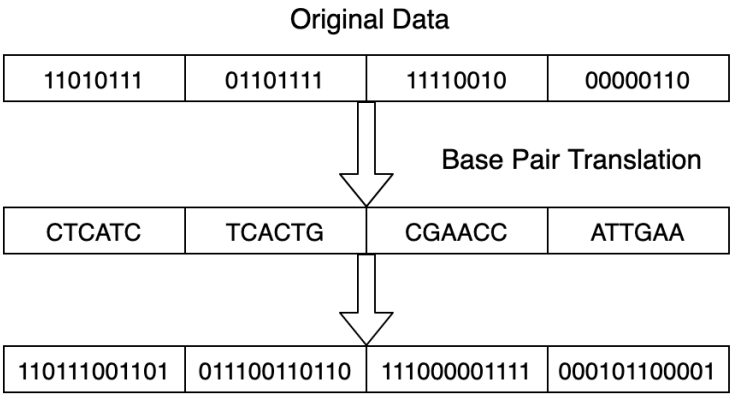

Figure 4.1: Example showing the encoding of 4 bytes of data into base pairs, followed by their binary equivalent.

Thus, the maximum bit efficiency of this mapping is 8/6, which is 1.33 bits/bp. There is the possibility of an 8 to 5 mapping if better primers are identified, however due to the extremely low error requirements, this seems unlikely at the present.

4.2 BCH Code Measurements

An analysis of the uncorrectable error rate of the BCH code is then done using equation 3.5. p_e^i is calculated using a set of values for synthesis errors of 1% down to 1x10^{-2}%, and sequencing errors are 1% and 0.1%. It is noted that these are treated as independent errors, but that there exists a possibility of a synthesis error that is corrected by a second sequencing error. Taking the ideal chain length of 1000 bps, valid values of t are defined in terms of the original byte symbol groupings, where $k + 2t = n$.

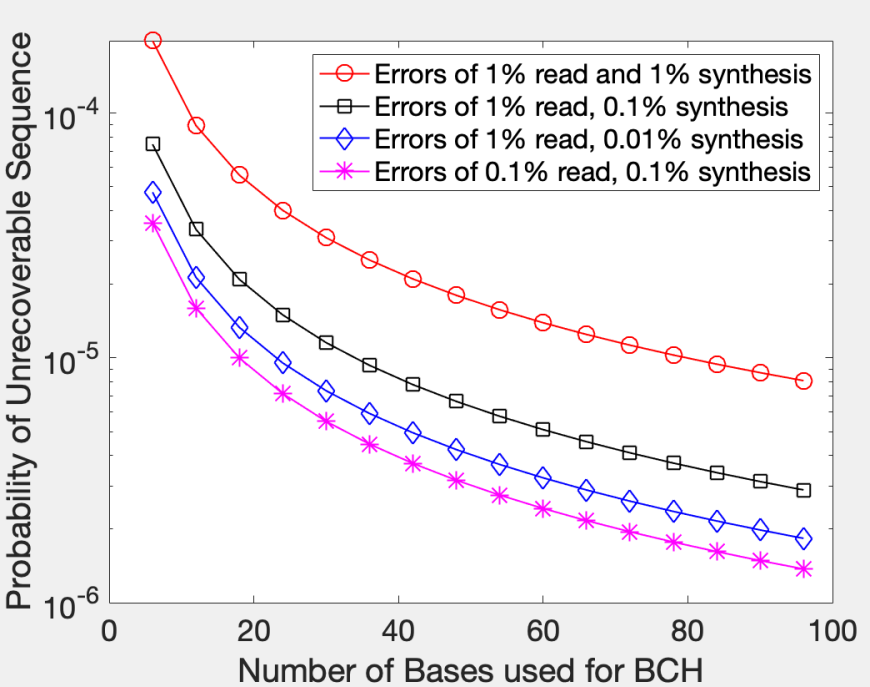

Figure 4.2: Plot showing probability of BCH unrecoverable failure versus the number of bases used for correction. In this case, the 1% read with 0.01% synthesis and 0.1% read with 0.1% synthesis appear to overlap, but the latter is instead slightly lower.

As expected, as the synthesis and sequencing go down, the probability of an error also go down. Given that BCH codes are well understood, none of this is new information, instead it is useful as a set of reasonable metrics for what is reasonable in a current DNA system. However, at the strand level, increasing the number of correctable bits reduces the overall encoding efficiency. The following shows the error probability versus net encoding efficiency of individual strands.

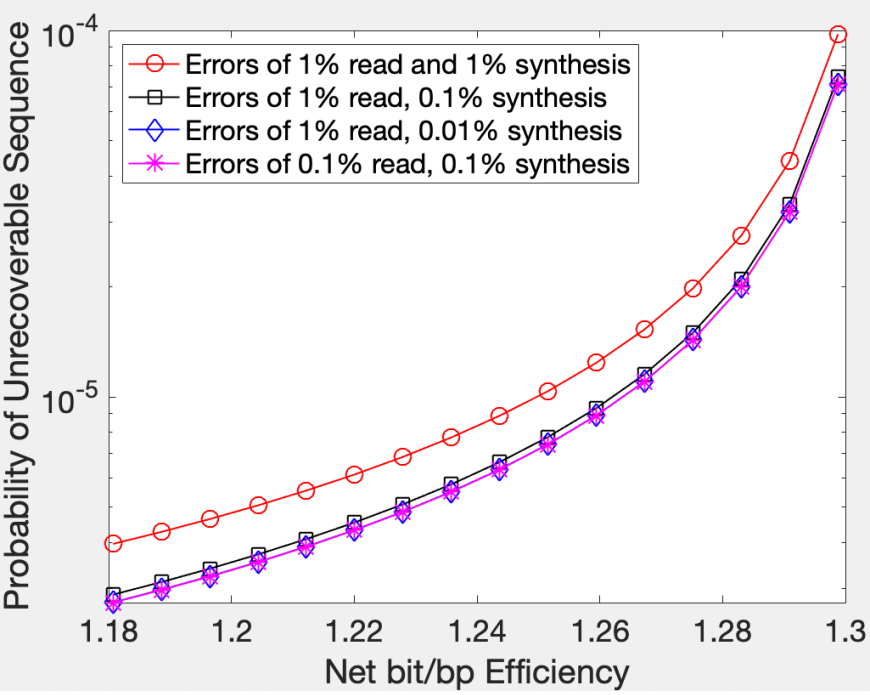

Figure 4.3: Plot providing a comparison between the error probability and next efficiency. In this case it is clear that the maximum allowable error rates are much lower than what a large-scale system currently reasonably can provide.

To hit the desired case of under 1% unrecoverable rate at a reasonable efficiency of 1.28 bits/bp, both synthesis and sequencing error rates need to be consistently below 0.1%. In particular, the upper bound of the traditional 6σ semiconductor bound must be below this limit to ensure memory integrity.

4.3 Analysis of Complete Raptor Code

This is then substituted into 3.6 as P_E^i. By adjusting the number of Raptor symbols in each strand, N_s, the number of Source Symbols, SS, and parity symbols, PS can be determined. The first measurement was a comparison between the probability of an unrecoverable pool with respect to the net bit/pb efficiency. This was performed on the four previously mentioned cases, where minimal Raptor codes were chosen, $SS + PS = 800$, leading to a balance of $SS = 784$, $PS = 16$, signifying a 2% overhead.

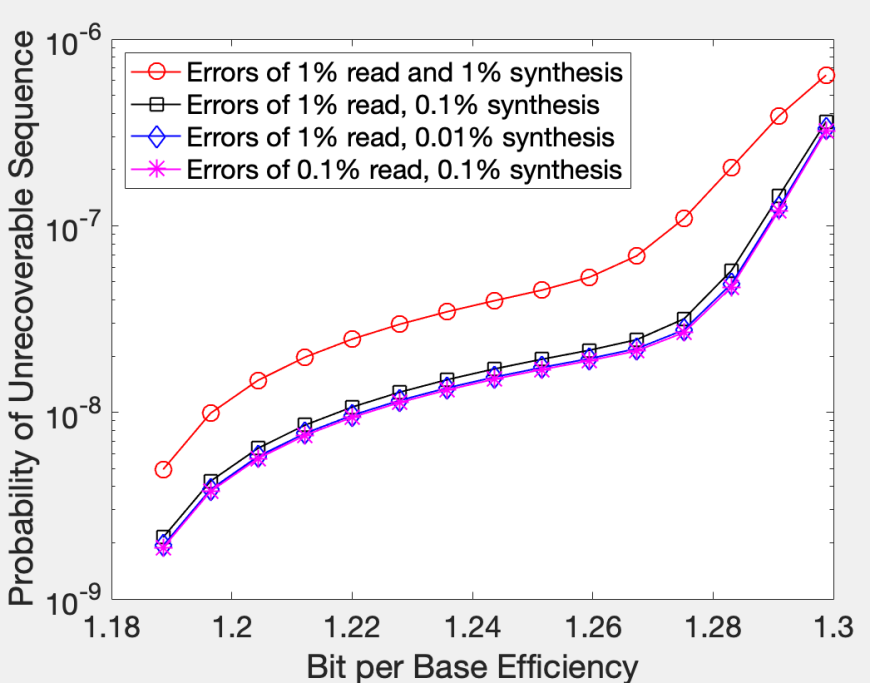

Figure 4.4: Plot showing the addition of a 2% Raptor overhead with $N_s = 1$.

In this case, The case of 0.1% read error and 0.1% sequencing error was chosen as a reasonable metric. The important case to note is the gradual bunching of values as the BCH error rates decrease. Additionally, in this plot it is easily seen where the two separate error recovery drive metrics are dominant. At this point, these errors are still too large for the target of 10^{-12}, so the Raptor overhead was increased.

The overhead was then varied, and the results are shown below. As expected, as the overhead increased, the probability of an unrecoverable sequence went down and it is possible to determine a reasonable cutoff for error values.

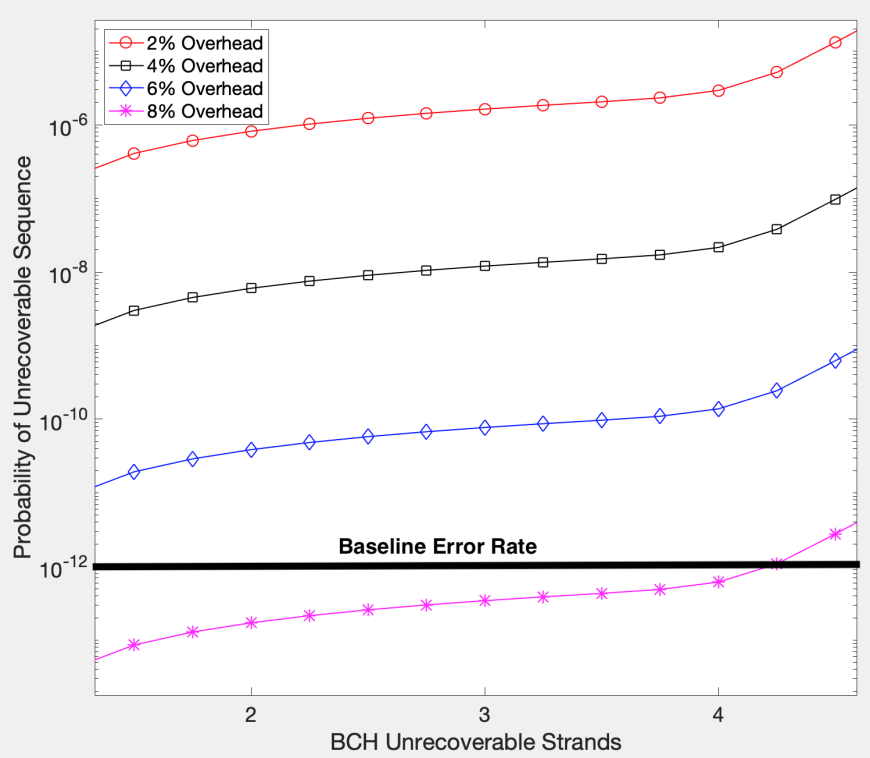

Figure 4.5: Plot showing error probability based on Raptor Overhead in terms of number of uncorrectable BCH errors.

From this, a family of curves is generated which allows for the extraction of required overhead in terms of BCH strand errors . Thus, if the sequencer and synthesis tools have a known error rate, the required overhead can be found to match the target error rate. For this example, a total net efficiency of 1.18 bits/bp was found to satisfy the 10⁻12 error rate requirement.

4.4 An Efficiency Comparison of Encoding Techniques

Following this, it is then possible to compare these results with some of the other proposed techniques and highlight some assumptions that were made in various works that might skew the results.

Table 4.3: A table comparing selected works to the data generated in this work.

	Church et al.[25]	Goldman et al[10].	Bornholt et al.[42]	Erlich et al.[57]	This Work
Theoretical Density [b/bp]	1.0	1.58	1.58	1.98	1.33
Redundancy	None	4	1.5	1.07	1.08
Error Correction	None	Repetition	RS	RS	BCH
Net Efficiency [b/bp]	0.83	0.33	0.88	1.57	1.18

In particular, this chart is unable to properly compare the overall recovery data, which makes this work look worse by comparison. In particular, the net efficiency is greatly reduced to ensure data integrity, while some of the earlier works focused on showing that the recovery was possible at all. Additionally, the significant advancements made by Erlich et al. target the reading of a single pool, rather than focusing on individual strand addressability. This provides better results, but comes at a cost of increased sequencing time and a non-linear error correction method.

This does however highlight the need to transition to an 8 to 5 encoding scheme, which would boost the theoretical density and net efficiency significantly. As error rates are reduced and sequencing methods improve, this becomes more and more possible, particularly if it becomes possible to increase the strand size, thus reducing the number of unique primers. Future work should target a better method of primer development for arbitrary data sequences, because there is a major gap in research in this sector.

Chapter 5

A Practical System Understanding

Following the development of a system outline as well as the encoding method, the basic construction of a full system becomes obvious. Unfortunately, while the main recovery flow presented in 3.3 and 3.3 is relatively straightforward to implement, the error flow makes this much more complicated. This chapter outlines a potential error recovery flow which makes use of the more precise primer addressing method to perform targeted read-retries. These presented flows were then verified using an Arria V FPGA to ensure that these are hardware implementable.

5.1 A Practical Understanding of Error Flow

One of the most important considerations in this entire work is the concept of read-retries, specifically when a sequence is unrecoverable. Unlike other memory, where read settings such as voltage level can be adjusted to properly recover data, DNA sequencing doesn't provide a parallel that can be leveraged off of. PCR then needs to be selectively leveraged to not only amplify the original data, but also to perform selective amplification on addresses determined by the microcontroller. Thus, if the microcontroller is able to detect that a sequence is unrecoverable due to a BCH indicated read failure, it should be able to make a second read request to . However, this isn't nearly as quantifiable due to dropout restrictions and the sequencer dependence. Below is a state diagram of a method to perform a read of a specific pool.

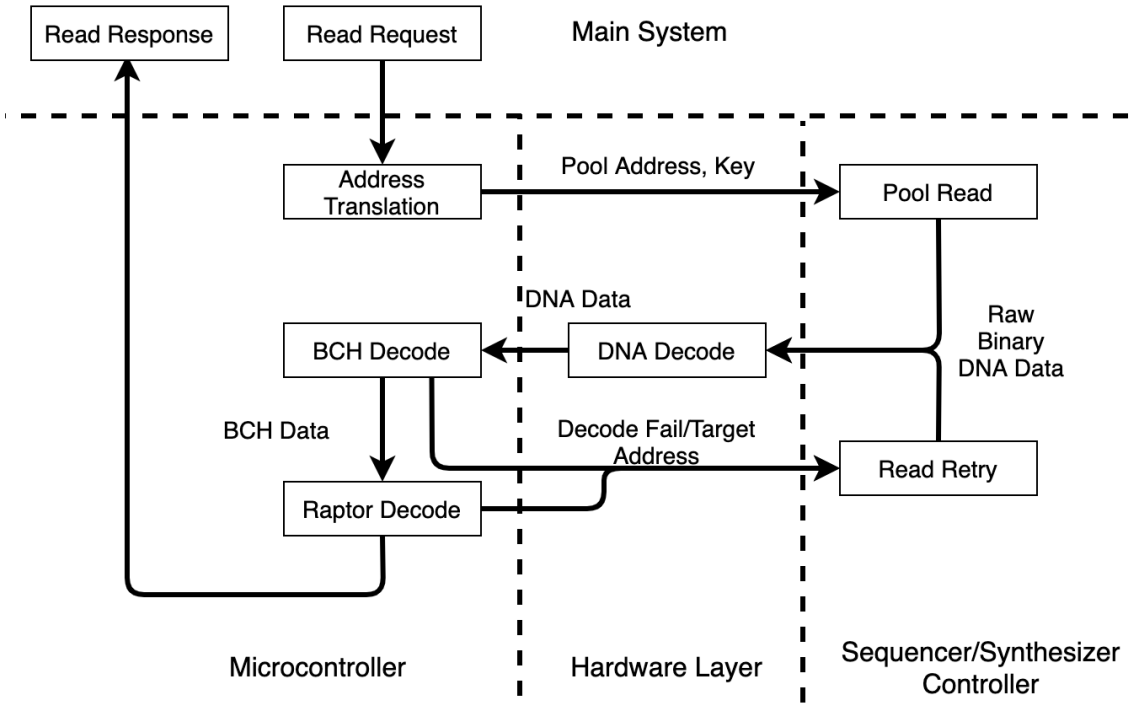

Figure 5.1: Figure showing a potential design for DNA Memory error recovery. The logic controlling the BCH Decode and the Raptor decode are responsible for holding the correction table to ensure that retries are performed on the correct strands.

Taking this system and defining the amount of time to perform a normal read as t_r and targeted read as t_{tr}, a loose approximation for operation can be developed. Using the probability of a certain number of failed pages from before in conjunction with this, an expected read-retry holdup time can then be determined using the following equation. Other calculation time is trivial in comparison, being more than 4 orders of magnitude below. The key metric in this case is the ability to perform parallel sequencing, thus allowing multiple read attempts at the same time. However, this adds an extra overhead of requiring additional DNA movement.

$$t_{read} = \begin{cases} t_r, N_{BCHValid} >= K \\ t_r + \frac{\sum_i^{NumMissingSymbols} t_{tr} P_{se,i(E_i<t)}}{parallelreads}, N_{BCHValid} < K \end{cases} \tag{5.1}$$

Furthermore, using some limiting number of single address retries, r, allows for a probability of a mis-read correction, shown below. By tweaking this parameter, it is

then possible to improve actual reads and lower the apparent sequencer error rate, assuming the sequence was encoded correctly in the first place. This provides clear evidence that the synthesis error rate is much more important than the sequencing error rate due to the ability to correct for errors at an exponentially improved rate.

$$P_{BCHError} = (1 - \sum_{i=0}^{t} \binom{n}{i} p_{se}^{i} (1 - p_{se})^{n-i})^{r} \tag{5.2}$$

5.2 In-System Division of Tasks

One of the key points largely ignored to this point is the split between the microcontroller, the hardware layer, and the DNA system itself. A proposed system breakdown is given in Figure 5.2, which is heavily inspired by existing SSD and HDD designs. Due to the nature of the memory, it sits in a middle-ground between a disk drive and flash memory due to the required memory movement, but at a much smaller scale. While HDDs are completely reliant on the internal controller, DNA memory is able to offload some translation logic onto hardware without significant timing penalty. However, given that a much more complicated scheme of error correction is required, the entire error flow must be handled on the microcontroller, rather than division between the microcontroller and on-die logical elements.

Other points to note with this division is that the microcontroller will need significant off-controller storage in the form of flash memory as well as a NOR storage table equivalent. The flash memory in this case is to store larger data files before they are translated, to reduce system overhead by allowing fast write operations of smaller files. The NOR memory is then used as a faster alternative to storing. Given the immense storage size that DNA systems will control, NOR memory might not be the ideal solution, so this could be moved entirely to flash at a latency cost.

Controlled by Microcontroller	Controlled by Physical Logic	Controlled by Microfluidic System
Logical Address Translation Address Table Storage Flash Temporary Storage BCH/Raptor Error Correction BCH/Raptor Encoding Read-Retry Requests	Address to Pool Translation Primer Key Decoding Binary to Pool-Binary	Pool Movement Interface with Synthesizer/Sequencer Base to Base-Binary Translation

Figure 5.2: Proposed breakdown of tasks between the on-board microcontroller, the physical hardware, and the microfluidic system. The microfluidic system in this case can either be on the board itself, causing it to be grouped with the physical logic, or a separate off-system controller in the DNA storage system.

The physical logic portion of this table was then implemented in an FPGA design using VHDL to act as a proof of concept. While BCH can be implemented directly onto hardware designs, which would reduce overall microcontroller overhead, it provides a marginal advantage due to the existing read length overhead. This isn't a problem until read times fall far enough to no longer be the limiting factor, or if the ability to perform parallel sequencing dramatically increases throughput.

5.3 Microfluidic Structural Concerns

The final section of discussion is to address one of the more unique requirements of the microfluidic control structure. One of the key problems with this memory is the issue of actually performing a read operation, given the number of pools compared to the number of theoretical read elements. The use of microfluidic storage lends itself easily to a 3d structure, but the following depiction portrays a number of different potential microfluidic pool to sequence flows in a 2d depiction. The target of this diagram is to show various architecture designs, particularly the specific pool to sequencing pool designs versus a more open architecture, which would allow pools to be read at any available read region.

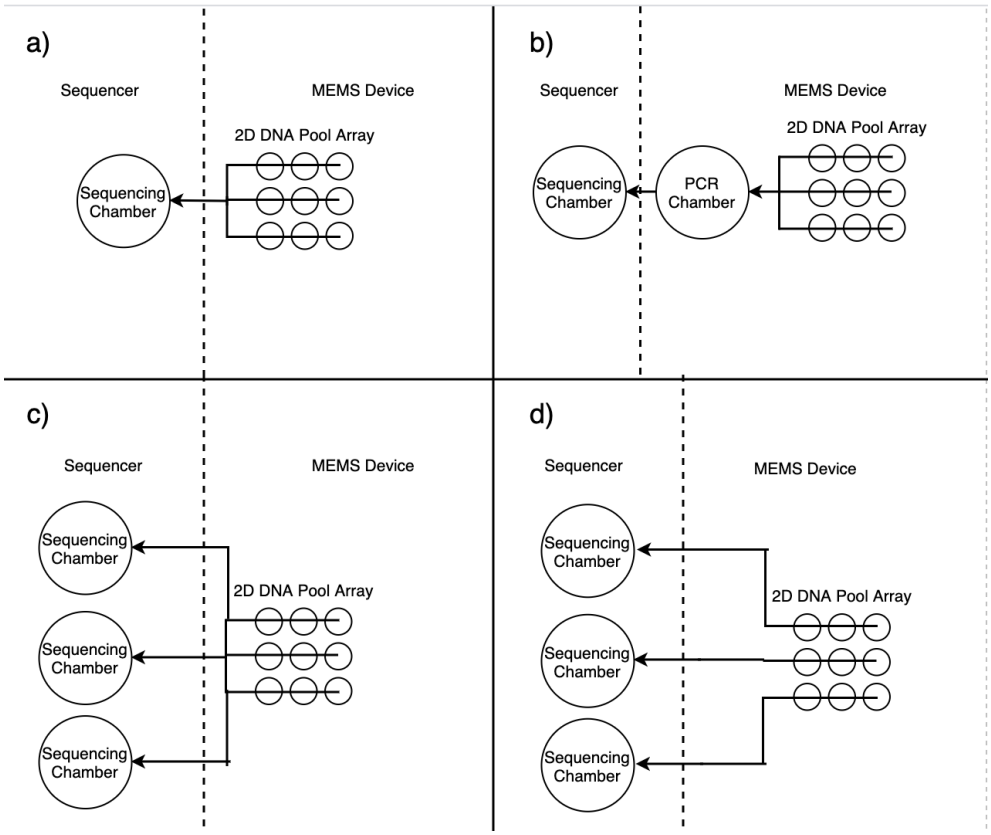

Figure 5.3: Image of four possible methods of pool read physical layouts. A) shows a basic layout where all the pools are connected to the external sequencer, where B) the PCR region has been moved inside the MEMS device to reduce sequencer labor. C) and D) then depict two methods of of parallel sequencing methods, with either complete read ability or specific region read locality.

Each of these proposed designs has various drawbacks, but the general trade off is between the ability for concurrent reads and microfluidic complexity. While the goal would be to perform as many concurrent reads as possible, this is currently limited to the sequencing technology, but eventually to the ability for the transport mechanisms to reach the read chambers. Unlike HDDs or tape, where the read head is highly mobile, DNA systems currently would be required to efficiently move individual pools to the reading location, which increases the risk of data loss. Future work should begin to develop physical versions of these microfluidic structures to verify that such a design is possible. While this work is unable to properly answer many of these questions, they will likely be the subject of many future works.

Chapter 6

Concluding Remarks

In this work, the concepts and applications of DNA memory were developed, along with an understanding of technological issues and pitfalls. Encoding methods for high distributed error sequences were investigated, and a modified Raptor with a BCH pre-code was analyzed. A binary to base conversion method was developed and the entire system was combined to calculate the expected error rates.

For a practical example with a pool of 800 strands and 1000 bp per strand, a net efficiency of 1.18 bits per bp was found to successfully recover from errors. This was then successfully compared with the existing literature, demonstrating that this is a feasible memory solution able to fail less than 1 in 10^{-12} pool reads. Portions of this proposed encoding method were then successfully translated into a hardware design on an FPGA, demonstrating a practical example made possible by the linear complexity of the Raptor algorithm.

There is still significant work before DNA memory becomes a common form of archival storage, but the architectural groundwork has been developed to the point that progress is dependent on the advancement of microfluidics and DNA sequencing methods. The application in deep archival storage makes this an important step as humanity moves into the century of atom, bit, and gene.

Bibliography

[1] P. Atkinson, "Computer memories: The history of computer form," *History and Technology*, vol. 15, pp. 89–120, 09 1998.

[2] F. Masuoka, M. Momodomi, Y. Iwata, and R. Shirota, "New ultra high density eprom and flash eeprom with nand structure cell," vol. 33, pp. 552 – 555, 02 1987.

[3] Y. Cai, "Errors in flash-memory-based solid-state drives," 2017.

[4] IDC, "where in the world is storage 2013," 2013.

[5] Y.-S. L. R. S. MAKALA, J. Alsmeier, "Three dimensional nand device and method of charge trap layer separation and floating gate formation in the nand device," U.S. Patent 8658499B2, July 2012.

[6] J. Clark, "Amazon launches glacier cloud storage, hopes enterprise will go cold on tape use," Aug 2012.

[7] G. Rozenberg, B. Thomas, and J. Kok, *Handbook of natural computing*. Springer, 2012.

[8] A. M. e. a. E., "The half-life of dna in bone: measuring decay kinetics in 158 dated fossils," Oct 2012.

[9] G. M. Church, Y. Gao, and S. Kosuri, "Next-generation digital information storage in dna," Sep 2012.

[10] N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. M. LeProust, B. Sipos, and E. Birney, "Towards practical, high-capacity, low-maintenance information storage in synthesized dna," Jan 2013.

[11] "Darpa molecular data storage project to handle information flood," Feb 2019.

[12] T. M. T. Kgil, D. Roberts, "Improving nand flash based disk caches," June 2008.

[13] J. J. Schwartz and S. R. Quake, "Single molecule measurement of the "speed limit" of dna polymerase," Dec 2009.

[14] J. J. Schwartz, "Correction for schwartz et al., single molecule measurement of the 'speed limit' of dna polymerase," Jan 2010.

[15] A. Ahmadian, B. Gharizadeh, A. C. Gustafsson, F. Sterky, P. Nyrén, M. Uhlén, and J. Lundeberg, "Single-nucleotide polymorphism analysis by pyrosequencing," *Analytical Biochemistry*, vol. 280, no. 1, p. 103–110, 2000.

[16] T. Abdullah, M. Faiza, P. Pant, M. R. Akhtar, and P. Pant, "An analysis of single nucleotide substitution in genetic codons - probabilities and outcomes," *Bioinformation*, vol. 12, no. 3, p. 98–104, 2016.

[17] S. J. Balin and M. Cascalho, "The rate of mutation of a single gene," *Nucleic Acids Research*, vol. 38, p. 1575–1582, Sep 2009.

[18] W. S. et. al, "Evolution of the insertion-deletion mutation rate across the tree of life," *Genes, Genomes, Genetics*, June 2016.

[19] D. Bikard, C. Loot, Z. Baharoglu, and D. Mazel, "Folded dna in action: Hairpin formation and biological functions in prokaryotes," *Microbiology and Molecular Biology Reviews*, vol. 74, no. 4, p. 570–588, 2010.

[20] S. Kosuri and G. M. Church, "Large-scale de novo dna synthesis: technologies and applications," *Nature Methods*, vol. 11, no. 5, p. 499–507, 2014.

[21] R. A. Hughes and A. D. Ellington, "Synthetic dna synthesis and assembly: Putting the synthetic in synthetic biology," *Cold Spring Harbor Perspectives in Biology*, vol. 9, no. 1, 2017.

[22] C.-C. Lee, T. M. Snyder, and S. R. Quake, "A microfluidic oligonucleotide synthesizer," *Nucleic Acids Research*, vol. 38, no. 8, p. 2514–2521, 2010.

[23] J. L. Weber and E. W. Myers, "Human whole-genome shotgun sequencing," *Genome Research*, vol. 7, pp. 401–409, Jan 1997.

[24] M. Gauthier, *Simulation of polymer translocation through small channels: A molecular dynamics study and a new Monte Carlo approach.* PhD thesis, 09 2007.

[25] J. A. Shendure, G. J. Porreca, and G. M. Church, "Overview of dna sequencing strategies," *Current Protocols in Molecular Biology*, vol. 81, no. 1, 2008.

[26] "Dna sequencing and next-generation sequencing methods," *Modern Biophysical Chemistry*, p. 285–302, 2014.

[27] H. Lee, J. Gurtowski, S. Yoo, M. Nattestad, S. Marcus, S. Goodwin, W. R. Mccombie, and M. C. Schatz, "Third-generation sequencing and the future of genomics," 2016.

[28] A. Srinivasa, J. Tani, K. Saboda, J. Borowsky, G. Ruvkun, M. Zuber, and C. Carr, "Development of a nucleic acid-based life detection instrument testbed," 03 2019.

[29] A. e. a. Chien, "Deoxyribonucleic acid polymerase from the extreme thermophile thermus aquaticus," *Journal of bacteriology*, vol. 127,3, pp. 1550–7, 1976.

[30] L. Pray, "The biotechnology revolution: Pcr and the use of reverse transcriptase to clone expressed genes," 2008.

[31] M. Willsey et. al, "Puddle: A dynamic, error-correcting, full-stack microfluidics platform," *ASPLOS 2019*, p. 183, Apr 2019.

[32] H. e. a. Lee, "Terminator-free template-independent enzymatic dna synthesis for digital information storage.," *Nature Communications*, 2019.

[33] J. Barr, "amazon-ebs-update-new-cold-storage-and-throughput-options," Apr 2016.

[34] "Hamr technology," tech. rep., Seagate, 2018.

[35] M. Lantz, "why-the-future-of-data-storage-is-still-magnetic-tape," August 2018.

[36] S.-I. Iwasaki, "Perpendicular magnetic recording—its development and realization," *Journal of Magnetism and Magnetic Materials*, vol. 324, no. 3, p. 244–247, 2012.

[37] P. Sharma, Q. Zhang, D. Sando, C. H. Lei, Y. Liu, J. Li, V. Nagarajan, and J. Seidel, "Nonvolatile ferroelectric domain wall memory," *Science Advances*, vol. 3, no. 6, 2017.

[38] T. Eshita, T. Tamura, and Y. Arimoto, "Ferroelectric random access memory (fram) devices," *Advances in Non-Volatile Memory and Storage Technology*, p. 434–454, 2014.

[39] C. Tanaka, K. Abe, H. Noguchi, K. Nomura, K. Ikegami, and S. Fujita, "A scaling of cell area with perpendicular stt-mram cells as an embedded memory," *2014 14th Annual Non-Volatile Memory Technology Symposium (NVMTS)*, 2014.

[40] S. Bhatti, R. Sbiaa, A. Hirohata, H. Ohno, S. Fukami, and S. Piramanayagam, "Spintronics based random access memory: a review," *Materials Today*, vol. 20, no. 9, p. 530–548, 2017.

[41] X. Dong, N. P. Jouppi, and Y. Xie, "Pcramsim," *Proceedings of the 2009 International Conference on Computer-Aided Design - ICCAD 09*, 2009.

[42] J. Bornholt, R. Lopez, D. Carmean, L. Ceze, G. Seelig, and K. Strauss, "A dna-based archival storage system," *IEEE Micro*, p. 1–1, 2017.

[43] R. E. Fontana, R. G. Biskeborn, M. Lantz, and G. M. Decad, "Tape in the cloud—technology developments and roadmaps supporting 80 tb cartridge capacities," *AIP Advances*, vol. 9, p. 125222, Jan 2019.

[44] L. Adleman, "Molecular computation of solutions to combinatorial problems," *Science*, vol. 266, p. 1021–1024, Nov 1994.

[45] M. Ogihara and A. Ray, "Simulating boolean circuits on a dna computer," *Algorithmica*, vol. 25, no. 2-3, p. 239–250, 1999.

[46] Y. Benenson, B. Gil, U. Ben-Dor, R. Adar, and E. Shapiro, "An autonomous molecular computer for logical control of gene expression," *Nature*, vol. 429, no. 6990, p. 423–429, 2004.

[47] M. Barbaro, A. Bonfiglio, L. Raffo, A. Alessandrini, P. Facci, and I. Barák, "Fully electronic dna hybridization detection by a standard cmos biochip," *Sensors and Actuators B: Chemical*, vol. 118, no. 1-2, p. 41–46, 2006.

[48] S. Sarkar, "Decoding "coding"- information and dna," *BioScience*, vol. 46, p. 857–864, Dec 1996.

[49] D. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, p. 1098–1101, 1952.

[50] S. Singh, *The code book: the science of secrecy from ancient Egypt to quantum cryptography*. Distributed by Paw Prints, Baker and Taylor, 2009.

[51] M. Ailenberg and O. D. Rotstein, "An improved huffman coding method for archiving text, images, and music characters in dna," *BioTechniques*, vol. 47, no. 3, p. 747–754, 2009.

[52] N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. M. Leproust, B. Sipos, and E. Birney, "Towards practical, high-capacity, low-maintenance information storage in synthesized dna," *Nature*, vol. 494, no. 7435, p. 77–80, 2013.

[53] S. Reed and G. Solomon, "polynomial codes over certain finite fields," *Society of Industrial Applied Mathematics*, vol. 8, Jun 1960.

[54] R. Bose and D. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Information and Control*, vol. 3, no. 1, p. 68–79, 1960.

[55] A. Gehani, T. Labean, and J. Reif, "Dna-based cryptography," *DNA Based Computers V DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, p. 233–249, 2000.

[56] J. Bornholt, R. Lopez, D. Carmean, L. Ceze, G. Seelig, and K. Strauss, "A dna-based archival storage system," *IEEE Micro*, p. 1–1, 2017.

[57] Y. Erlich and D. Zielinski, "Dna fountain enables a robust and efficient storage architecture," *Science*, vol. 355, p. 950–954, Feb 2017.

[58] C. N. Takahashi, B. H. Nguyen, K. Strauss, and L. Ceze, "Demonstration of end-to-end automation of dna data storage," *Scientific Reports*, vol. 9, no. 1, 2019.

[59] K. Chen, J. Zhu, F. Boskovic, and U. F. Keyser, "Secure data storage on dna hard drives," 2019.

[60] Y. Benjamini and T. P. Speed, "Summarizing and correcting the gc content bias in high-throughput sequencing," *Nucleic Acids Research*, vol. 40, Sep 2012.

[61] W. Feng, D. Xue, F. Song, D. Chen, Z. Li, and X. Wang, "A method for homopolymer length discrimination in ion torrent sequencing," *2015 34th Chinese Control Conference (CCC)*, 2015.

[62] K. Nybo, "Dna and general pcr methods: Pcr primer design," *BioTechniques*, vol. 46, no. 7, p. 505–507, 2009.

[63] M. C. F. Thomsen, H. Hasman, H. Westh, H. Kaya, and O. Lund, "Rucs: rapid identification of pcr primers for unique core sequences," *Bioinformatics*, vol. 33, no. 24, p. 3917–3921, 2017.

[64] T. C. Lorenz, "Polymerase chain reaction: Basic protocol plus troubleshooting and optimization strategies," *Journal of Visualized Experiments*, no. 63, 2012.

[65] M. Luby, "Lt codes," in *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pp. 271–280, 2002.

[66] A. Shokrollahi, "Raptor codes," *2007 IEEE Information Theory Workshop on Information Theory for Wireless Networks*, 2007.

[67] A. Shokrollahi, "Raptor codes," *International Symposium onInformation Theory, 2004. ISIT 2004. Proceedings.*

[68] G. Yu and J. Moon, "Concatenated raptor codes in nand flash memory," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, p. 857–869, 2014.

[69] C. Ong and C. Leung, "On the undetected error probability of triple-error-correcting bch codes," *IEEE Transactions on Information Theory*, vol. 37, no. 3, p. 673–678, 1991.

[70] M.-G. Kim and J. H. Lee, "Undetected error probabilities of binary primitive bch codes for both error correction and detection," *IEEE Transactions on Communications*, vol. 44, no. 5, p. 575–580, 1996.

Appendices

Appendix A: Full Table for DNA Encoding

Table A.1: Full chart showing the translation between the binary values and the base pair with the binary encoding.

Binary	Base Encoding	Mapped Encoding	Binary	Base Encoding	Mapped Encoding	Binary	Base Encoding	Mapped Encoding	Binary	Base Encoding	Mapped Encoding
0b0	ATAAAG	000100000010	0b1000000	ACCGAT	001111100001	0b10000000	TAACTG	010000110110	0b11000000	GTTTGA	100101011000
0b1	ATAGAA	000100100000	0b1000001	ACCGAG	001111100010	0b10000001	TAACTC	010000110111	0b11000001	CACACT	110011001101
0b10	ATAGAT	000100100001	0b1000010	TGTTTG	011001010110	0b10000010	TAAATT	010000000101	0b11000010	CACTCC	110011011111
0b11	ATAGAG	000100100010	0b1000011	TGTGTT	011001100101	0b10000011	TAAATG	010000000110	0b11000011	CACTCT	110011011101
0b100	ATTAAA	000101000000	0b1000100	TGTGTG	011001100110	0b10000100	GCGGGC	101110101011	0b11000100	CACGCA	110011101100
0b101	ATTTAG	000101010010	0b1000101	TGTCTT	011001110101	0b10000101	GCGGGA	101110101000	0b11000101	CACGCT	110011101101
0b110	ATTGAA	000101100000	0b1000110	TGTCTG	011001110110	0b10000110	GCGCGG	101110111010	0b11000110	CAACCT	110000111101
0b111	ATTGAT	000101100001	0b1000111	TGTATT	011001000101	0b10000111	GCGAGG	101110001010	0b11000111	CAAACC	110000001111
0b1000	ATTGAG	000101100010	0b1001000	TGTATC	011001000111	0b10001000	GCGAGA	101110001000	0b11001000	CAAACT	110000001101
0b1001	ATTCAG	000101110010	0b1001001	TGGTTT	011010010101	0b10001001	GCGTGG	101110011010	0b11001001	CAATCC	110000011111
0b1010	ATGAAA	000110000000	0b1001010	TGGGTT	011010100101	0b10001010	GCGTGA	101110011000	0b11001010	CAATCT	110000011101
0b1011	ATGTAA	000110010000	0b1001011	TGGGTG	011010100110	0b10001011	GCCGGC	101111101011	0b11001011	CAAGCC	110000101111
0b1100	ATGTAT	000110010001	0b1001100	TGGGTC	011010100111	0b10001100	GCCCGA	101111111000	0b11001100	CAAGCA	110000101100
0b1101	ATGTAG	000110010010	0b1001101	TGGCTT	011010110101	0b10001101	GCCAGC	101111001011	0b11001101	CATACC	110001001111
0b1110	ATGGAA	000110100000	0b1001110	TGGCTG	011010110110	0b10001110	GCCTGC	101111011011	0b11001110	CATACA	110001001100
0b1111	ATGGAT	000110100001	0b1001111	TGGATC	011010000111	0b10001111	GCACGC	101100111011	0b11001111	CATACT	110001001101
0b10000	ATGCAA	000110110000	0b1010000	TGCTTT	011011010101	0b10010000	GCATGG	101100011010	0b11010000	CATTCT	110001011101
0b10001	ATGCAT	000110110001	0b1010001	TGCGTG	011011100110	0b10010001	GCATGC	101100011011	0b11010001	CATGCA	110001101100
0b10010	ATGCAG	000110110010	0b1010010	TGCGTC	011011100111	0b10010010	GCATGA	101100011000	0b11010010	CAGCCT	110010111101
0b10011	ATCTAG	000111010010	0b1010011	TGCATT	011011000101	0b10010011	GCTGGA	101101101000	0b11010011	CAGACT	110010001101
0b10100	ATCGAA	000111100000	0b1010100	TGCATG	011011000110	0b10010100	GCTAGA	101101001000	0b11010100	CAGTCC	110010011111
0b10101	ATCCAA	000111110000	0b1010101	TGATTT	011000010101	0b10010101	GCTTGC	101101011011	0b11010101	CTCCCA	110111111100
0b10110	ATCCAT	000111110001	0b1010110	TGAGTG	011000100110	0b10010110	GAGGGG	100010101010	0b11010110	CTCACA	110111001100
0b10111	AGAAAA	001000000000	0b1010111	TGAATG	011000000110	0b10010111	GAGGGA	100010101000	0b11010111	CTCACT	110111001101
0b11000	AGAAAT	001000000001	0b1011000	TGAATC	011000000111	0b10011000	GACGGG	100011101010	0b11011000	CTCTCT	110111011101
0b11001	AGAAAG	001000000010	0b1011001	TCTTTC	011101010111	0b10011001	GACGGC	100011101011	0b11011001	CTACCA	110100111100
0b11010	AGATAA	001000010000	0b1011010	TCTGTT	011101100101	0b10011010	GACCGG	100011111010	0b11011010	CTAACC	110100001111
0b11011	AGAGAA	001000100000	0b1011011	TCTCTT	011101110101	0b10011011	GACCGC	100011111011	0b11011011	CTAACA	110100001100
0b11100	AGACAA	001000110000	0b1011100	TCTCTG	011101110110	0b10011100	GACCGA	100011111000	0b11011100	CTATCA	110100011100
0b11101	AGTAAA	001001000000	0b1011101	TCTATT	011101000101	0b10011101	GACTGG	100011011010	0b11011101	CTATCT	110100011101
0b11110	AGTAAT	001001000001	0b1011110	TCTATG	011101000110	0b10011110	GACTGC	100011011011	0b11011110	CTAGCA	110100101100
0b11111	AGTGAA	001001100000	0b1011111	TCGTTG	011110010110	0b10011111	GACTGA	100011011000	0b11011111	CTTCCC	110101111111
0b100000	AGTCAT	001001110001	0b1100000	TCGGTT	011110100101	0b10100000	GAAGGA	100000101000	0b11100000	CTTACT	110101001101
0b100001	AGGAAT	001010000001	0b1100001	TCGGTG	011110100110	0b10100001	GAACGG	100000111010	0b11100001	CTTTCA	110101011100
0b100010	AGGAAG	001010000010	0b1100010	TCGCTT	011110110101	0b10100010	GAAAGC	100000001011	0b11100010	CTTTCT	110101011101
0b100011	AGGTAA	001010010000	0b1100011	TCGCTG	011110110110	0b10100011	GAATGG	100000011010	0b11100011	CTTGCA	110101101100
0b100100	AGGTAG	001010010010	0b1100100	TCGATT	011110000101	0b10100100	GATGGG	100001101010	0b11100100	CTTGCT	110101101101
0b100101	AGGGAA	001010100000	0b1100101	TCGATG	011110000110	0b10100101	GATCGG	100001111010	0b11100101	CTGCCA	110110111100
0b100110	AGGGAT	001010100001	0b1100110	TCCTTG	011111010110	0b10100110	GATAGA	100001001000	0b11100110	CTGCCT	110110111101
0b100111	AGGCAA	001010110000	0b1100111	TCCTTC	011111010111	0b10100111	GATTGG	100001011010	0b11100111	CTGACC	110110001111
0b101000	AGGCAT	001010110001	0b1101000	TCCGTT	011111100101	0b10101000	GTGGGA	100110101000	0b11101000	CTGACA	110110001100
0b101001	AGGCAG	001010110010	0b1101001	TCCGTC	011111100111	0b10101001	GTGAGC	100110001011	0b11101001	CTGGCC	110110101111
0b101010	AGCAAT	001011000001	0b1101010	TCCCTG	011111110110	0b10101010	GTGAGA	100110001000	0b11101010	CTGGCT	110110101101
0b101011	AGCAAG	001011000010	0b1101011	TCCATT	011111000101	0b10101011	GTGTGA	100110011000	0b11101011	CGCACA	111011001100
0b101100	AGCTAT	001011010001	0b1101100	TCATTG	011100010110	0b10101100	GTCGGG	100111101010	0b11101100	CGCACT	111011001101
0b101101	AGCTAG	001011010010	0b1101101	TCATTC	011100010111	0b10101101	GTCGGC	100111101011	0b11101101	CGCGCC	111011101111
0b101110	AGCGAT	001011100001	0b1101110	TCAGTG	011100100110	0b10101110	GTCGGA	100111101000	0b11101110	CGCGCA	111011101100
0b101111	AGCCAA	001011110000	0b1101111	TCACTG	011100110110	0b10101111	GTCCGC	100111111011	0b11101111	CGACCC	111000111111
0b110000	AGCCAG	001011110010	0b1110000	TCACTC	011100110111	0b10110000	GTCAGG	100111001010	0b11110000	CGACCA	111000111100
0b110001	ACATAG	001100010010	0b1110001	TCAATG	011100000110	0b10110001	GTCTGA	100111011000	0b11110001	CGACCT	111000111101
0b110010	ACAGAG	001100100010	0b1110010	TATTTC	010001010111	0b10110010	GTAGGG	100100101010	0b11110010	CGAACC	111000001111
0b110011	ACACAA	001100110000	0b1110011	TATGTT	010001100101	0b10110011	GTAGGC	100100101011	0b11110011	CGAACA	111000001100
0b110100	ACTTAA	001101010000	0b1110100	TATGTC	010001100111	0b10110100	GTACGG	100100111010	0b11110100	CGAACT	111000001101
0b110101	ACTGAA	001101100000	0b1110101	TATATT	010001000101	0b10110101	GTACGC	100100111011	0b11110101	CGATCC	111000011111
0b110110	ACTGAG	001101100010	0b1110110	TAGGTT	010010100101	0b10110110	GTAAGA	100100001000	0b11110110	CGATCA	111000011100
0b110111	ACTCAG	001101110010	0b1110111	TAGCTG	010010110110	0b10110111	GTATGG	100100011010	0b11110111	CGAGCA	111000101100
0b111000	ACGAAG	001110000010	0b1111000	TAGCTC	010010110111	0b10111000	GTATGC	100100011011	0b11111000	CGAGCT	111000101101
0b111001	ACGTAA	001110010000	0b1111001	TAGATT	010010000101	0b10111001	GTTGGC	100101101011	0b11111001	CGTACA	111001001100
0b111010	ACGTAG	001110010010	0b1111010	TACTTC	010011010111	0b10111010	GTTCGC	100101111011	0b11111010	CGTACT	111001001101
0b111011	ACGGAA	001110100000	0b1111011	TACGTT	010011100101	0b10111011	GTTCGA	100101111000	0b11111011	CGTTCA	111001011100
0b111100	ACGCAT	001110110001	0b1111100	TACCTG	010011110110	0b10111100	GTTAGG	100101001010	0b11111100	CGTGCC	111001101111
0b111101	ACGCAG	001110110010	0b1111101	TAATTC	010000010111	0b10111101	GTTAGC	100101001011	0b11111101	CGTGCT	111001101101
0b111110	ACCAAT	001111000001	0b1111110	TAAGTT	010000100101	0b10111110	GTTAGA	100101001000	0b11111110	CGGTCT	111010011101
0b111111	ACCGAA	001111100000	0b1111111	TAAGTG	010000100110	0b10111111	GTTTGG	100101011010	0b11111111	CGGGCC	111010101111